Masaryk University
Faculty of Informatics

Artificial Neural Network for Precipitation Nowcasting

Master's Thesis

Bc. Vladimíra Ježeľová

Brno, Fall 2018

Replace this page with a copy of the official signed thesis assignment and a copy of the Statement of an Author.

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Bc. Vladimíra Hežeľová

Advisor: RNDr. Roman Stoklasa, Ph.D.

Acknowledgements

I would like to thank my advisor RNDr. Roman Stoklasa, Ph.D., for his valuable help, pieces of advice, and willingness during the elaboration of this thesis.

I would also like to express my gratitude to my friends and family for all their support during the elaboration of this thesis and during my studies. This includes my boyfriend, Jaroslav Řehořka, for his advice regarding the thesis content, insights, patience, and support.

Finally, I thank the Czech Hydrometeorological Institute and the developers of Meteoradar application for making the radar data available with a favorable license.

Abstract

Precipitation nowcasting problem tackles an accurate forecast for the near future by extrapolation of weather radar images. Prediction of the next frame, that is capturing the movement and the development of rain is typically approached by techniques as numerical weather prediction or optical flow.

Recently, artificial neural networks overcome traditional approaches in many tasks, such as image classification or language translation. Therefore, we believe that the capabilities of neural networks could be exploited for weather forecasting as well.

In this thesis, we analyze the problem of precipitation nowcasting and provide an overview of related work. Ultimately, we propose a deep neural network Unitos4 that forms recurrent ConvLSTM layers into a structure similar to U-Net. It benefits from the ConvLSTM in capturing time dependencies between frames as well as from U-Net in extracting important features and global context along with precise localization. We show that Unitos4 outperforms the currently used method by the Czech Hydrometeorological Institute in all our evaluations.

Keywords

machine learning, neural network, weather forecasting, precipitation nowcasting, image processing, recurrent neural network, long short-term memory, convolutional neural network, convLSTM network

Contents

1 Introduction

Weather has a significant impact on people's everyday lives and plays an important role in many fields, such as aviation, marine, and agriculture. Extreme meteorological conditions directly influence many aspects of human lives, including human health and safety in case of natural disasters, such as severe thunderstorms, tropical cyclones, floods, etc.

The weather is the state of the atmosphere for a given time and place. It is typically described by a set of meteorological variables, which includes temperature, air pressure, wind, cloudiness, rainfalls, and more. Each of these variables is measured over time in many places around the country and around the world, so a global model of the weather can be estimated.

In this thesis, we will focus only on the precipitation aspect of weather.

Precipitation activity and locations of rain, showers, and thunderstorms are monitored in real-time by Doppler weather radars thanks to the fact, that water droplets reflect radio waves. This radar reflectivity is measured in dBZ[1] units. The reflectivity is also proportional to the rain intensity (measured in mm/h units) by Marshall-Palmer formula, so it can be easily derived.

Measured radar data are typically rendered over a 2D map, which is also known as CAPPI (Constant Altitude Plan Position Indicator) image. For illustration purposes, the measured reflectivity value is mapped to a corresponding color using a specific look-up table, so it is easy to distinguish the negligible rain from hailstones. Figure 1.1 illustrates such an image.

Because the weather influences our everyday lives, it would be very helpful to predict how it develops in the near (or further) future. This problem is addressed by weather forecasting.

There are several techniques for weather forecasting. However, all of the typical techniques are based on a numerical model of the atmosphere and are computed for a certain period of time. The weather forecasting problem is very complex since there are plenty of variables, which influences the real weather but cannot be measured or observed. Thus, many different techniques and models exist that tries to achieve better performance than others on a specific subset of tasks. Such well-known weather prediction models are, e.g., Aladin [2], Medard [3], WRF [4], etc. Each of them has its advantages and disadvantages. Typically, a different forecasting technique is suitable for a prediction of a different set of variables, like rainfall, tem-

1. dBZ - decibels of Z

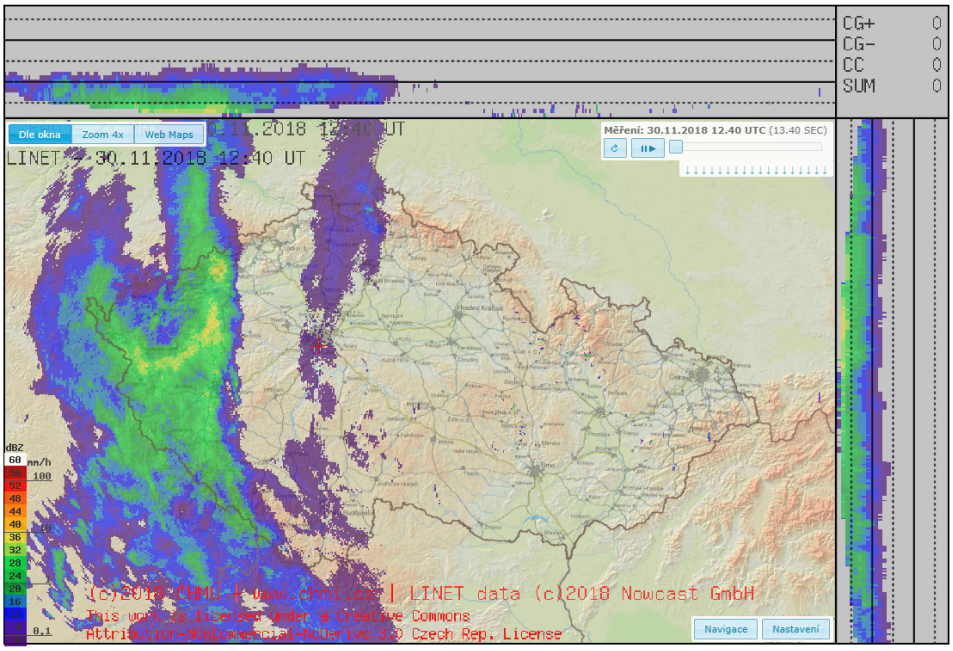


Figure 1.1: A screenshot of CHMI Nowcasting web-portal. It shows a CAPPI image, precipitation reflectivity rendered in colors over a map of the Czech Republic. The scale illustrates mapping a specific reflectivity [dbZ] to a corresponding color as well as to an estimated intensity [mm/h] *Source:* Czech Hydrometeorological Institute [1].

perature, cloudiness and so on. Some models are able to produce long-term predictions, while others are more suitable only for short-term ones.

Beside long-term and short-term forecasting methods, there is also nowcasting, a small-scale forecast to the near future. It produces a prediction only for the next few minutes or hours (0 - 6 hours) [5]. But on the other hand, it aims at precise localization of rain along with an accurate estimation of intensity.

In this thesis, we analyze the precipitation nowcasting problem, i.e., the prediction of the next radar image based on the known radar images from previous time-points. Our goal is to propose a new solution to predict future weather radar maps.

There are three different approaches that are typically used for precipitation nowcasting: Numerical Weather Prediction, techniques based on optical

flow and extrapolation, and machine learning approaches utilizing artificial neural networks.

Numerical weather prediction (NWP) [6] for precipitation forecasting had been proposed already in 1922. Although it has been improving since then, it is still not accurate and has many drawbacks. In particular, it requires simulations of atmospheric circulation and complex numerical equations that lead to a prediction error. Therefore, NWP is more suitable for long-term forecasts than short-term.

Techniques based on optical flow are often preferred over NWP, especially for short-term prediction. They can achieve higher precision and require less computational complexity. An approach based on the optical flow relies on remaining a certain pattern of movement in the atmosphere that is projected also into movement of "rain cells" in CAPPI images. Prediction by optical flow models performs a non-rigid transformation by preserving the direction and velocity of objects from a sequence of previous frames [7, 8]. However, precipitation nowcasting by optical flow has limits, since it takes into account a few previous frames and does not exploit information from similar historical weather situations. Because these types of techniques basically extrapolate the last known frame, they are not able to predict the internal development of showers and thunderstorms, whether the rain intensity is increasing, or decreasing over time. The most recent technique and still an active field of research are based on deep neural networks. They have already achieved a considerable success in many areas, such as image classification, speech recognition, language translation, and many others. On the other hand, there are still many problems they can solve, or where existing models can be improved. A few architectures have been already proposed for precipitation nowcasting [9, 10]. Nevertheless, they are confronted with the complexity of weather development and do not achieve accurate prediction.

We are interested in weather prediction in the area of the Czech Republic. The Czech Hydrometeorological Institute (CHMI) provides Nowcasting web-portal [1], which presents the precipitation situation for one past hour, current situation, and one hour of the forecast. Weather radar images are available in 10-minute intervals. We can observe that the prediction of extensive rain regions appears to be a non-rigid transformation that preserves the pattern of apparent motion. In addition, the forecast struggles with local precipitation with rapid development. Therefore, we assume that the forecasting method is based on the optical flow.

In this thesis, we analyze the possibilities for precipitation nowcasting, consider suitable alternatives, and present a new solution based on deep

neural networks. Our proposed neural network outperformed the current method for nowcasting in the Czech Republic.

In Chapter 2, we analyze the problem of nowcasting, divide precipitation into three main types and discuss their characteristics and appearance on CAPPI images. Further, we provide a brief overview of related work. In Chapter 3, we explain neural network architecture that is related to the problem od next frame prediction. Later on, in Chapter 4, we present our proposed architecture Unitos4 that combines ConvLSTM network and U-Net. In Chapter 5, we describe dataset used for training, validation, and testing as well as all preprocessing steps. Then, we discuss implementation part including training in Chapter 6. Finally, in Chapter 7, we present experiments, evaluation, results, and comparison with CHMI prediction.

2 Analyses of Precipitation Nowcasting

Accurate prediction of precipitation intensity in conjunction with its precise location is a challenging task. This chapter will analyze the problem of accurate precipitation nowcasting and provides an overview of related work.

In Section 2.1 will be reviewed main precipitation types. For each type, first, its standard behavior and characteristics will be discussed, second how it appears in a weather radar image, and lastly how it affects forecasting.

Numerical weather prediction (NWP), optical flow based methods, and prediction models based on neural networks are three major approaches to precipitation nowcasting problem. Section 2.2 summarizes relevant research works for these approaches.

2.1 Precipitation Nowcasting Problem

Precipitations can be divided into three main types: convective, frontal, and orographic.

Convective rain is caused by convective ascend of an air mass due to its higher temperature comparing to its surrounding [11]. This phenomenon yields significant cores with higher reflectivity. In the weather radar map, it corresponds to a small region with high intensity. Convective rain is often identified by cell structures, instability, and the lifetime of particular rain cells is only about dozens of minutes [12]. Optical flow struggles particularly with this type of precipitation. The main reason is the rapid development of the convective rain. On the other hand, forecast by the neural network could benefit from information obtained from similar historical data situations.

Frontal precipitation is caused by air ascending along or near a weather front [11]. This precipitation tends to spread over an extensive area. Typically it does not have significant gradients of reflectivity; therefore, it appears in the radar map more uniform in contrast to convective rain. In addition, frontal precipitation lasts considerably longer, typically several hours [12]. For this type, the optical flow algorithm is more suitable and more accurate regarding the relatively stable direction of movement and more gradual development of the precipitation. Due to these factors, the frontal rain prediction could be less problematic for the neural network too.

Orographic precipitation is caused by the ascent of moist air over orographic barriers. As a result, rain shadow phenomenon can be seen on a leeward side. It is a region on the lee side of the mountain, where the precipitation is less frequent than on the windward side [11]. Fundamental optical

flow is not able to capture this phenomenon since it performs only a transformation preserving a movement pattern regardless of similar extraordinary situations occurring on certain locations repeatedly. We assume that the neural network could learn this effect on the specific areas of mountains, particularly in the case that the network input would be position-invariant to a map.

A suitable approach for precipitation nowcasting should be able to distinguish among these precipitations somehow to be able to behave accordingly.

2.2 Related Work

Numerical weather prediction (NWP), optical flow based methods, or neural networks can be used for precipitation nowcasting.

NWP models [6] are less suitable for short-term forecasting since they require complex simulations of the atmospheric circulation, computation of complex numerical equations, and high computational cost. That can result in prediction error in nowcasting due to a rapid development of weather. Under the circumstances, optical flow based methods can achieve more accurate prediction. Although they have limitations too and their prediction accuracy can be also increased, they are preferred in precipitation nowcasting systems.

One of the algorithms for nowcasting based on optical flow, presented in [7], overcomes an existing forecast method based on traditional advection schemes. Nonetheless, the improvement is little and the proposed algorithm has difficulties with prediction when the boundary of rain area are not moving simultaneously.

Precipitation nowcasting algorithm named ROVER[1] has been proposed in [8] for the Hong Kong Observatory (HKO). The research aims at improving the estimation of the flow speed of the existing algorithm deployed at that time in the HKO system for nowcasting. Although the ROVER algorithm has overcome former algorithms and consequently has replaced them, it requires empirical settings of parameters and the accuracy of prediction can still be increased. An overview of nowcasting algorithms, applications, and services in the HKO including the ROVER algorithm is provided in [13].

Convolutional Long Shor-Term Memory (ConvLSTM) Network for nowcasting problem has been introduced in [9]. The proposed architecture is a recurrent network with LSTM units. The ConvLSTM network has convolutional input-hidden and hidden-hidden connections in comparison to fully-

1. ROVER = Real-time Optical flow by Variational methods for Echoes of Radar

connected connections of a traditional LSTM network. Although the proposed model has achieved better results than both the ROVER algorithm and the fully connected LSTM network, the outcome of the network exhibits blurriness.

The more recent research [10] by the same authors presents two architectures along with benchmark for precipitation nowcasting. First, Convolutional Gated Recurrent Unit (ConvGRU) architecture is presented. It is a recurrent network with GRU units in comparison to ConvLSTM network consisting of LSTM units. Secondly, Trajectory Gated Recurrent Unit (TrajGRU) network is proposed. TrajGRU architecture has overcome ConvGRU architecture as well as the ROVER algorithm. The advantage of TrajGRU over ConvGRU is dynamically determined recurrent connections that can learn the location-variant structure.

Next frame video prediction, as well as the precipitation nowcasting, deals with a motion prediction. Several feature learning strategies for future image prediction from the video have been proposed in [14] along with a benchmark of next frame video prediction methods. The proposed strategies are a multi-scale architecture, gradient difference loss function, and next frame prediction training of Generative Adversarial Network (GAN) [15]. GAN is a dual-learning mechanism consisting of "generator" and "discriminator" network being trained at the same time that can be used for various problems. A result of the three proposed strategies is a model able to predict sharp future frames. The predictions are shown on a few video sequences available at [16].

Even though similar neural network architectures can be used for both precipitation nowcasting and video prediction, in nowcasting, the nature of precipitation movement and development, that has different characteristics from a flow of objects in next frame prediction, should be considered.

3 Neural Network Architectures

In this chapter, we provide the necessary insights into neural network architectures that are related to precipitation nowcasting problem. First, we discuss convolutional neural networks that are used for tasks, such as image classification or object detection. Second, we explain recurrent neural networks that are able to learn time dependencies. Finally, we describe architectures that process image sequences and are used for next frame prediction.

3.1 Convolutional Neural Networks

Convolutional neural network (CNN, ConvNet) is considered to be a state-of-the-art image recognition system. A significant breakthrough in ConvNet was the research on hand-written digit recognition conducted by Yann LeCun et al. The proposed LeNet-5 architecture achieved 1 % error rate and about a 9 % reject rate on MNIST dataset (1998) [17]. Later on, Alex Krizhevsky et al. designed AlexNet, a deep convolutional neural network for image classification, trained on ImageNet dataset containing 1.3 million images. They outperformed all previous approaches with 39.7 % top-1 and 18.9 % top-5 error rate (2012) [18]. Since then, many improvements and modifications of convolutional neural network have been proposed. Later in this chapter, we explain U-Net and ConvLSTM architectures, both based on ConvNet, but meantime, we describe the basic concept of ConvNet.

Neural network typically consists of one input layer, one or more hidden layers, and one output layer. In the traditional fully-connected neural network (FCNN), each neuron of a layer is connected to each neuron of the previous layer. This results in a tremendous number of connections between the layers, which are referred to as weights. Therefore, it is impossible to create a deep fully-connected neural network consisting of many layers due to immense complexity and performance limitation. However, a neural network capable of learning complex structures requires deeper architecture. The ConvNet tackle this issue by sharing weights. Figure 3.1 compares the architectures of these two networks [19].

ConvNet employs mathematical operation convolution (3.1) instead of general matrix multiplication in at least one layer. The convolution is denoted by $*$ symbol. In the context of ConvNet, the input layer (input image) is convolved with the kernel (filter) represented by neural network weights, and the output of the convolution represents the subsequent hidden layer,

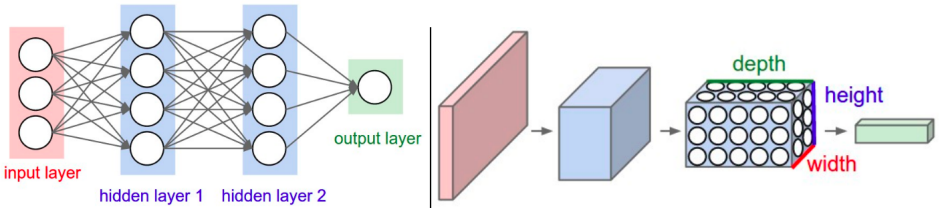


Figure 3.1: *Left:* A **fully-connected neural network**. *Right:* A **convolutional neural network** arranges its neurons in 3 dimensions, where depth represents feature maps. The pink input layer represents an input image. Special dimensions are reduced as a result of convolution [19].

called feature map or activation map. The input layer, instead of the input image, can be also a feature map. Then, the convolution layer transform one feature map of low-level features (edges, corners) to another feature map of high-level features (parts of object). The feature map consists of neurons, as well as the fully-connected hidden layer [19].

The convolution is defined in the following equation, where I is 2-dimensional image and K is 2-dimension kernel [20]. By analogy, we can get convolution for either one or three dimensions. Example of 2D convolution is depicted in Figure 3.2.

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{3.1}$$

The use of convolution in neural network is based on the assumption that the feature which is important at some position in the image, it is with high probability relevant also in other positions in that image. Therefore, weights can be shared across the whole image. In addition, kernel size 3×3 is often sufficient for recognition of the relevant features. As a result, sharing small kernels of weights over the whole image or feature map allow the network to represent the features with a significantly smaller number of weights and thus with less training parameters [20].

The convolution layer consists of several feature maps that are stacked one after another, each along with the respective kernel. The kernel represents certain feature and activation of a neuron in the feature map expresses the probability of the feature occurrence at the connected region in the image, called receptive field [20].

The convolution layer is typically followed by a pooling layer which is essentially subsampling. Typically used max pooling overlaps input image

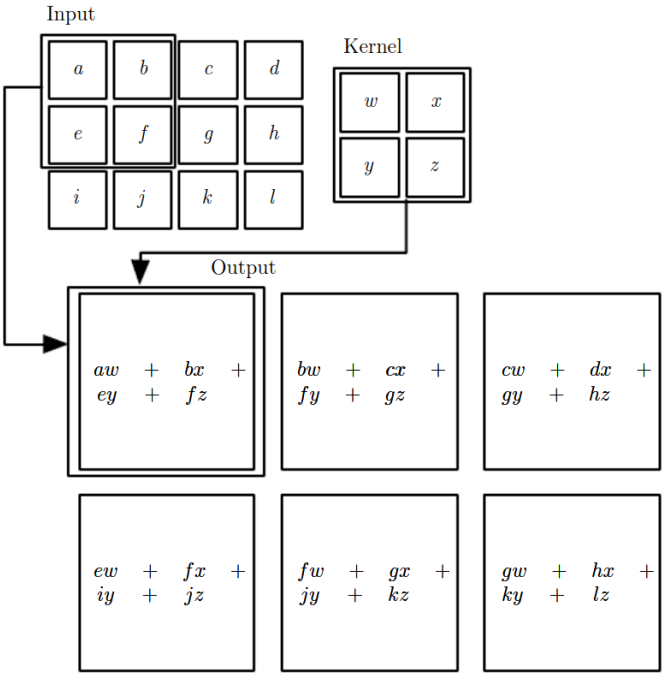


Figure 3.2: An example of 2D **convolution** without kernel flipping. [20]

with a region $x \times y$ px, typically 2×2 px, and produce the maximum value from the overlapping region to the output. This region is shifted with a certain stride over the entire image. This increases translation and rotation invariance, increases generalization, and reduces overfitting of the network [20].

There are typically one or more fully connected (FC) layers at the end of the architecture, which are equivalent to the FC layers in FCNN.

3.1.1 U-Net

Fundamental convolutional networks that stack several convolutional and pooling layers, one after another, are suitable especially for classification tasks. In this case, the repeated subsampling by a pooling layer is not a complication because the input dimension can be gradually decreased to a final prediction, which is normally only one class. However, image prediction, unlike class prediction, deals with the problem of precise localization. For the output image, each pixel needs to be classified.

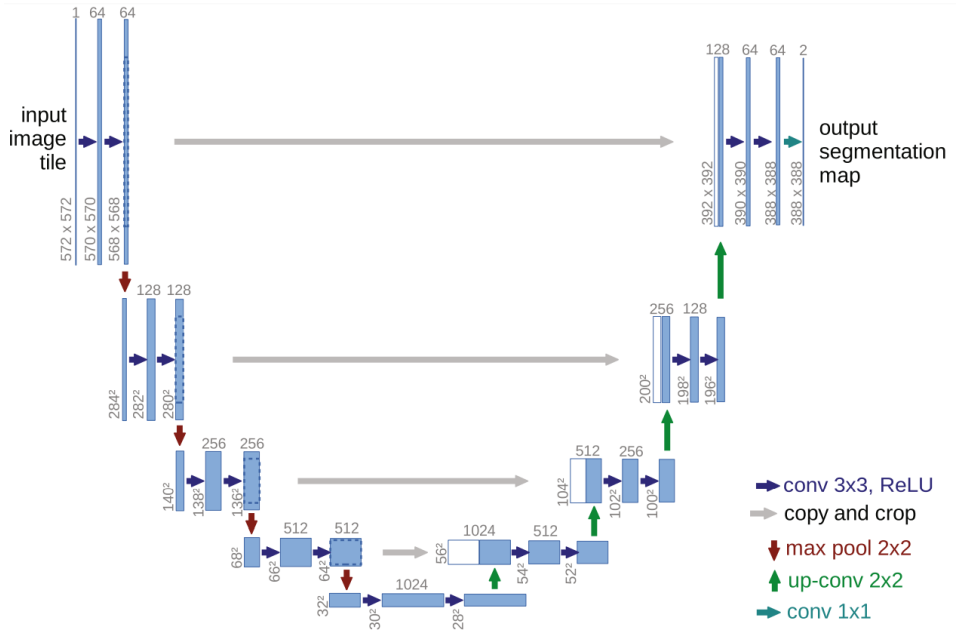


Figure 3.3: **U-Net**: a convolutional neural network for biomedical image segmentation. [21]

U-Net [21] is a convolutional network that addresses the problem of precise localization. It was designed for biomedical image segmentation by Ronnenberg et al. and won the ISBI cell tracking challenge 2015. The model consists of a contracting and an expanding path shown in Figure 3.3. The pooling layers in the contracting path enable to capture important features, global context, and reduce noise. Subsequently, expanding path due to upsampling, and connections to corresponding layers from contracting path, enables precise localization.

3.2 Recurrent Neural Networks

Recurrent neural network (RNN) process sequential data, such as words in a text, frames in a video, and so on. While ConvNet shares weights over the image representing space, RNN shares weights across the time. One timestep consists of one input, such as a word for speech recognition or a frame for the future frame prediction.

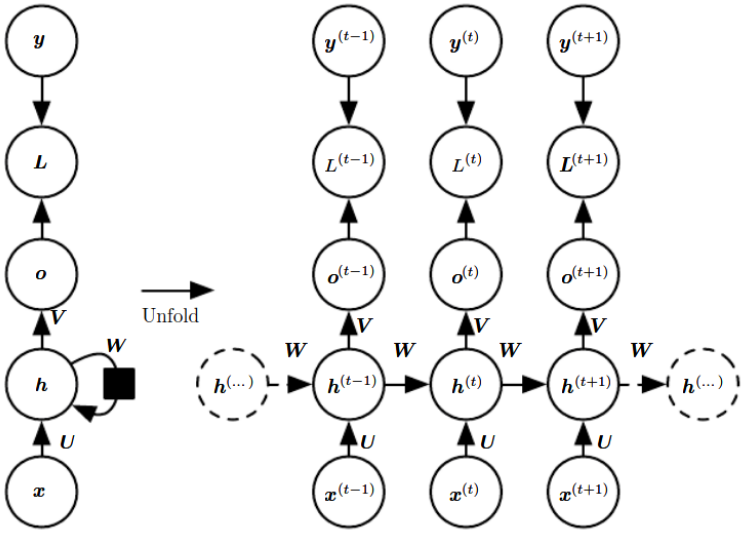


Figure 3.4: An example of a **recurrent neural network** with input x, hidden state h, output o, and target y. The loss L is computed as a distance of the output from the target. Input-to-hidden connections are represented by weight matrix U, hidden-to-hidden by W, and hidden-to-output by V. The black square symbolizes time delay. *Left:* A recurrent neural network and its loss drawn with recurrent connections. *Right:* The same seen as unfolded, where each node corresponds to one time-step [20].

RNN is characterized by recurrent connections in hidden layers. It has a so-called hidden state that stores information about previous elements of a time series. Such recurrent neural network is shown in Figure 3.4.

There are several types of RNN depending on input-output structure visualized in Figure 3.5. The suitable structure depends on the given task. For instance, one to many architecture is used for image captioning, many to one for sentiment classification, or many to many for language translation [19].

Vanilla RNN mentioned above encountered the vanishing gradient problem due to the repeated multiplication of the recurrent weight matrix by a derivative of the non-linear activation function. Let us suppose that the activation function is a commonly used hyperbolic tangent "tanh", with the range of values between -1 and 1; consequently, the range of derivative of this function is between 0 and 1 (Figure 3.6). Consequently, multiplying of the weight matrix by this derivative in backpropagation process results in

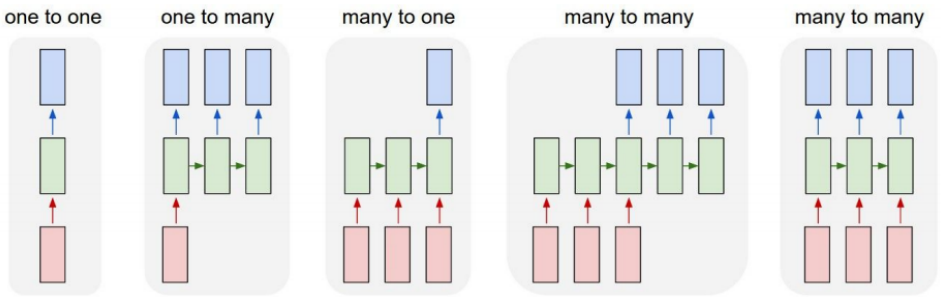


Figure 3.5: Different types of a recurrent neural network architecture. Input layer is displayed in red, hidden layer in green, and output layer in blue [19].

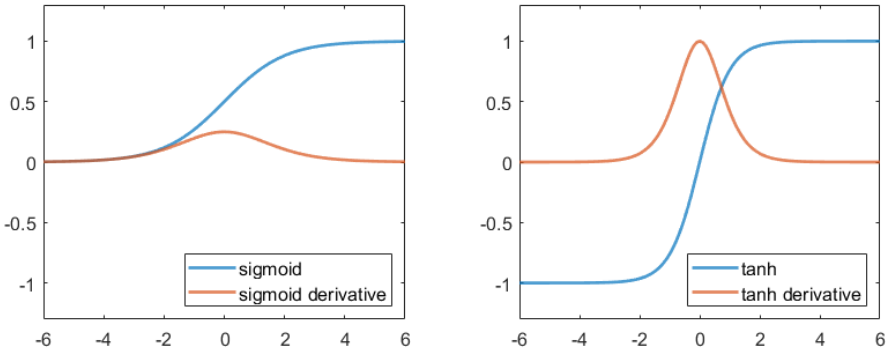


Figure 3.6: Activation functions **tanh** and **sigmoid**.

squeezing its range of values into the interval $(0, 1)$. Therefore, the derivatives of the loss function at time t, with respect to the activation functions, decreases exponentially as t increases [22].

To sum up, despite the considerable loss between ground truth and the prediction, the recurrent network might not be updating its weights, which means it is not learning anything relevant. This is a consequence of sharing weights across time-steps along with aforementioned nature of backpropagation.

To alleviate this problem, Hochreiter et al. [23] have proposed a gradient-based algorithm that enforces constant (not vanishing) propagation of loss through internal states of so-called **long short-term memory** (LSTM) **units**. It enables to learn long-term dependencies, dependencies from distant time-steps. Although there are several modifications of the LSTM unit, the one

shown in (3.2) is based on convLSTM unit, which we will discuss later in Section 3.3.

One LSTM unit corresponds to one time-step. An input of the unit in time t is denoted as x_t. The internal state of the unit is represented by hidden state h_{t-1} and cell state c_{t-1} from a previous cell, which corresponds to previous time-step. W_x are weight metrices connecting input with LSTM unit, and W_h representing recurrent connections. Activation functions sigmoid σ and $tanh$ are visualized in Figure 3.6. Biases for corresponding gates stand for b_i, b_f, b_g, and b_o. In addition, there are four gates: input gate i, forget gate f, gate g, and output gate o. While g_t represents new values that could be added to the internal state, input gate i_t determines how much to update each value of g_t. The outputs of these two gates are multiplied together. Symbol $\circ$ denotes the Hadamard product (element-wise multiplication). Meanwhile, forget gate f_t controls the level of cell state c_{t-1} reset how much of memory to forget. Consequently, sum of this new information and filtered "memory" is new cell state c_t. This new cell state is put through $tanh$ to adjust range of values between -1 and 1. Output gate o_t decides how much to release cell c_t. And the new cell state c_t and hidden state h_t flow to next cell $t+1$. These gates result in constant error backpropagation within memory cells; therefore, they allow to learn long-term dependencies. These dependencies are a significant advantage of LSTM compared to vanilla RNN.

$$
\begin{aligned}
i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \\
f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \\
g_t &= tanh(W_{xg}x_t + W_{hg}h_{t-1} + b_g) \\
o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \\
c_t &= f_t \circ c_{t-1} + i_t \circ g_t \\
h_t &= o_t \circ tanh(c_t)
\end{aligned}
\tag{3.2}
$$

An RNN network consisting of LSTM units is also called **long short-term memory** (LSTM) **neural network**.

3.3 Architectures for Image Sequence Processing

Precipitation nowcasting by a neural network, which is essentially the next frame prediction, requires an architecture that is able to process image sequence to capture spatial as well as time dependencies. We have already mentioned the relevant architectures, ConvLSTM network and GAN, in Section 2.2. In this section, we explain them more in detail.

Convolutional LSTM (ConvLSTM) model [9] is essentially LSTM network with convolutional connections between input to hidden layer and hidden to hidden layer instead of fully connected as they are in traditional LSTM network. ConvLSTM is more efficient for sequential image processing than fully-connected LSTM due to convolution connections. In (3.3) is shown Keras [24] implementation of convLSTM unit [9] gates and states.

$$\begin{aligned} i_t &= \sigma(W_{xi} * x_t + W_{hi} * h_{t-1} + b_i) \\ f_t &= \sigma(W_{xf} * x_t + W_{hf} * h_{t-1} + b_f) \\ g_t &= tanh(W_{xg} * x_t + W_{hg} * h_{t-1} + b_g) \\ o_t &= \sigma(W_{xo} * x_t + W_{ho} * h_{t-1} + b_o) \\ c_t &= f_t \circ c_{t-1} + i_t \circ g_t \\ h_t &= o_t \circ tanh(c_t) \end{aligned} \tag{3.3}$$

Another model that can be used for next image prediction is **Generative Adversarial Network** (GAN) [15]. It consists of two models: "generator" and "discriminator". The generator generates "artificial" images based on the data distribution. Consequently, the discriminator receives these images along with real images and is trained to distinguish between them. In the same time, the generator is trained to generate more realistic images until discriminator is not able to discriminate real from artificial. GAN in the context of next frame prediction can be used in the way, that the discriminator takes a sequence of images, and is trained to determine if the next frame is the real next frame or "artificial" from the generator. After training, the generator should be able to accurately predict the future image [25].

Either ConvLSTM model or GAN model can be used for next image prediction. Shi et al. in [9] with their ConvLSTM implementation have encountered the problem of blurriness of the predicted image. While GAN can predict sharp images [25]. On the other hand, GAN can be problematic to train due to its instability, as a result of two networks being trained at the same time [19].

4 Our Method

In this chapter, we present our approach to a problem of precipitation nowcasting, previously analyzed in Chapter 2. First, we explain our motivation to approach the problem by ConvLSTM architecture, outline our method in general and divide the problem into sub-problems. Later, in Section 4.2 we discuss elementary architecture consisting of ConvLSTM layers and explain its deficiencies. Ultimately, in the last section of this chapter, we present our final architecture Unitos4 that eliminates the main deficiencies of Primitos3.

4.1 Our Method in General

We have already discussed main approaches for precipitation nowcasting problem and our motivation to explore neural networks for that. As mentioned before, the trained network should be able to predict a future CAPPI image from a sequence of preceding images. That includes processing of time sequences, for which are recurrent networks suitable. However, elementary recurrent networks encounter the problem of vanishing gradients; therefore LSTM networks have been introduced to tackle this problem using LSTM cells. As a result, they are able to learn time dependencies farther away from each other. That is an important advantage of LSTM networks, as the predicted output is often influenced not only by recent but also by more distant past.

Although fully-connected LSTM networks are robust in the processing of natural language, in image processing, reducing the number of connections (training parameters) is necessary due to large dimensions of images. In recurrent networks for image processing, this problem is even more relevant. In deep neural networks, convolution has achieved excellent results in the extraction of important features in images. Thanks to a combination of LSTM cells and convolution operation, ConvLSTM is able to predict next frame in sequence based on extracting features using convolution (input–hidden connections) together with updating hidden state in individual timesteps (recurrent hidden–hidden connections). Therefore, we have decided to implement a recurrent ConvLSTM network and analyze its capabilities for next frame prediction. Moreover, ConvLSTM achieved decent results for prediction of future CAPPI image in the paper where this architecture was proposed.

The problem has been analyzed on three main precipitation types: convective, frontal, and orographic in Chapter 2. They are distinguished mainly

by the shape and size of individual rain cells, typical intensity structure, length of their lifetimes and velocity of their movement. Therefore, we have divided the problem into the following parts:

1. Movement of rain cells
   (a) Direction
   (b) Velocity
2. Development within individual rain cells – intensity changes

We first have focused on the problem of network capability to distinguish between various directions. Our goal was to train the network so that the predicted image would preserve the direction of the movement of objects from the input sequence. Additionally, for accurate prediction of CAPPI image, the network should be able to continue in the movement of objects in any direction.

The problem of velocity prediction can be in our case reduced to relevant velocities. Since we focus on the climate in the Czech Republic, we do not consider extreme velocities typical for tropical cyclones.

In addition to the accurate location of rain, we also strive for precise prediction of intensities. The network should be able to react according to the typical behavior of both frontal and convective precipitations as we discussed in Chapter 2. Their common development is similar in the following aspect. Both of them arise, then escalate and expand, and to the end weaken and reduce in size. That is the reason why the network should be able to learn statistics of the intensity curve in time. As a result, the network would accurately predict future development by intensity increase if the current location in the development curve is before the peak and analogously decrease if the current location is after the peak.

Besides convective and frontal types, the third type affected by orography does not occur in such extent in the Czech Republic. We assume that the invariant position of CAPPI images with respect to the map would be necessary for the accurate prediction of orographic precipitation. Since we prefer less complex architecture with a priority of better results for convective and frontal types, we have partitioned CAPPI images into smaller patches. Then, these patches are used as an input into the network in the same way regardless the global position of the patch in the map. Consequently, the output is again a patch of the same size, located at the same position on the map. By this approach, we do not have the ambition to be able to predict phenomena such as rain shadow or increased precipitation probability on the windward side of mountains.

Image partitioning into patches is convenient to fit the network into memory. An alternative possibility would be to subsample images; however, in such case, the precise localization would be lost. With patches, we are preserving original resolution and allow precise localization. Taking in the account already mentioned decreased complexity, the network has less training parameters, is less difficult to train, and the training is accelerated. The prediction is computed at each patch individually. To produce whole weather radar map it would be possible to merge all relevant patches in a suitable way. For instance, the patches could be partially overlapping and averaging of values that would correspond to each other in a global coordinate system would mitigate sharp transitions at the edge of each two adjacent patches. Nevertheless, the primary objective of this thesis is to analyze the problem and propose a solution for precipitation nowcasting, that can be done on the level of patches.

There are recurrent networks that are independent of input length. However, we do not see any benefit to having the variant length of input sequences in the case of our problem. Therefore, we have specified a uniform length of input sequences at the beginning of our research and all our experiments are based on that. To choose an unsuitable length could have a negative impact on network learning. On the one hand, an extremely small number would not capture enough information for accurate prediction, on the other hand, an excessively large number may take into account irrelevant historical information. Based on our observation of data, we have decided for input length of 9 frames. It corresponds to 90 minutes since the images are available in 10-minute intervals. For the output, we are interested in only one frame. The reason is that within this thesis we want to explore the possibilities and capabilities of neural networks for prediction of next CAPPI image. By prediction of several future time-steps error would be with high probability accumulated. In that case, the ability of the network itself would be obscured.

In the rest of this chapter, we present our two architectures with codenames *Primitos3* and *Unitos4*.

4.2 Initial Architecture: Primitos3

Our initial architecture, named **Primitos3**, consists of three ConvLSTM layers. First, we have focused on movement tracking and have observed that ConvLSTM network exhibits difficulties to capture faster movement.

Examples of predictions of randomly generated squares moving in a random direction are shown in Figure 4.1. We have created three datasets, each

of them consisting of 200 sequences, with 90 % for training and 10 % for validation. The first dataset contained sequences with a velocity of movement 3 px in one time-step, the second dataset with 6 px, and the third with 9 px. Primitos3 has been trained on 100 epochs for each dataset with the same hyperparameters. The network has predicted the next frame for the 3 px-speed almost accurately, while for the 6 px-speed little worse, and with the 9 px-speed had the most significant difficulties. We can see from the figure that the prediction of movement highly depends on the velocity. That is caused by small convolutional kernels that consider only small receptive fields and this information is location-invariant in recurrent connections.

The problem of location-invariance in ConvLSTM networks is described also in the more recent research [10] of the same authors who first proposed ConvLSTM architecture. Furthermore, they review that ConvLSTM network in the original paper was evaluated on only 97 rainy days and by measures probability of detection (POD), false alarm ratio (FAR), and critical success index (CSI) only at the 0.5 mm/h rain intensity threshold.

To capture higher velocity by the Primitos, we have increased the kernel size from 3×3 gradually to 5×5, 7×7, up to 9×9. Results in the Figure 4.1 were computed on Primitos3 with 3×3, 5×5, and 7×7 kernels from first to the last layer. On the one hand, the Primitos with larger kernels has been able to predict faster motion better, but on the other hand, larger kernels correspond to more weights. Hence, the learning is more problematic and the network contains noise caused by a huge number of training parameters and no reduction. For this problem typically pooling layers are employed in deep convolutional networks. They decrease the amount of noise whereby increase generalization – reduce overfitting. In our experiments with Primitos4, where no pooling layers have been employed, we have also observed, that the validation loss for the models with larger kernels started to rise much sooner than for the smaller kernels while the training loss has been still decreasing. Such behavior can be a sign of overfitting. Moreover, pooling layers increase translation and rotation invariance that is beneficial to capture motion. However, we want to predict the output of the same resolution to that of the input; therefore we cannot subsample without upsampling. In the next section, we are presenting our final architecture Unitos4 that deals with this problem.

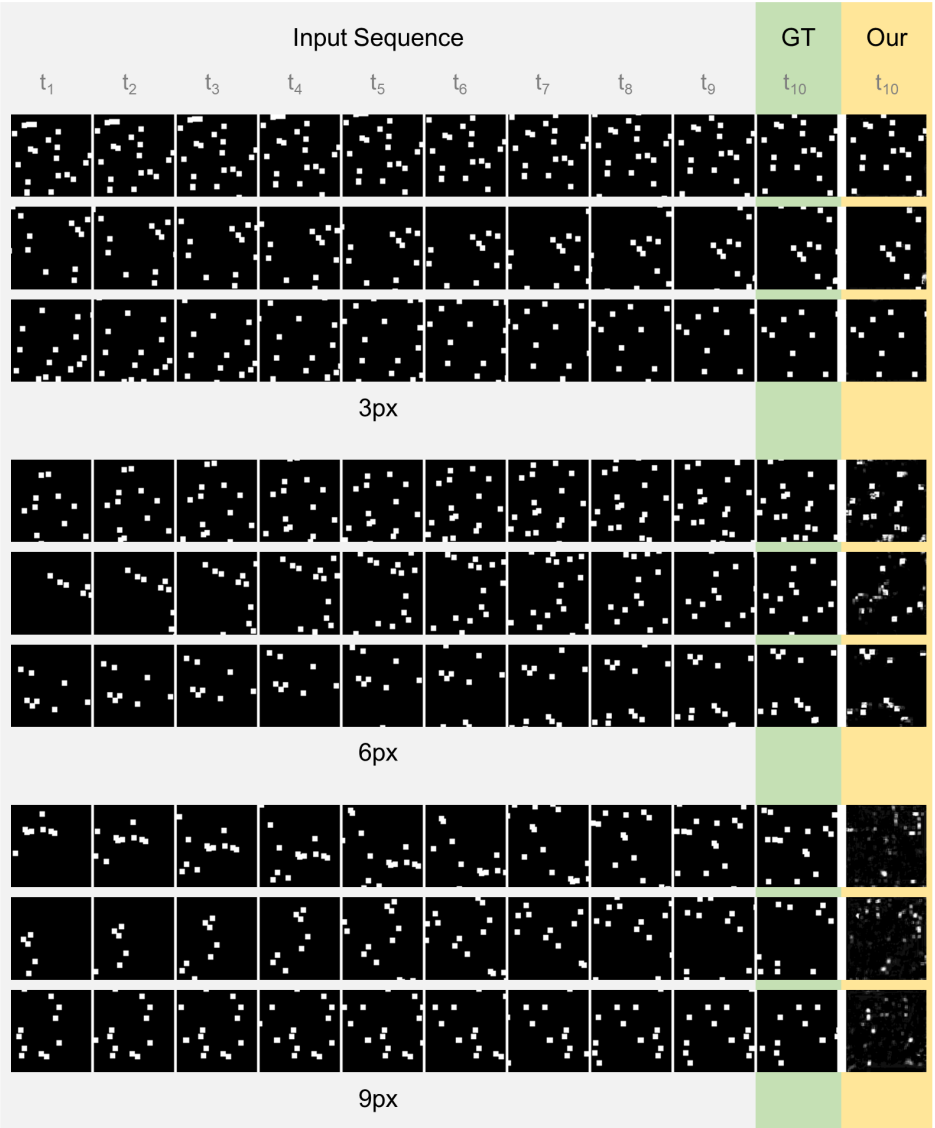


Figure 4.1: An experiment to explore the capability of vanilla ConvLSTM to capture more global context, a higher acceleration. *From up to down:* samples with randomly generated squares moving by 3 px, 6 px, and 9 px. Each speed contains three samples. *From left to right:* a sequence of 9 input frames from time t_1 to t_9, GT: ground truth at t_{10}, and OUR: our prediction at t_{10} from the input sequence.

4.3 Final Architecture: Unitos4

Vanilla ConvLSTM networks have the already discussed problem to capture more global context together with accurate localization. In the rest of this chapter, we propose an architecture that tackles this problem. Our proposed final architecture, named **Unitos4**, employs ConvLSTM layers similarly to Primitos4. However, in addition, the Unitos4 is formed in a structure inspired by U-Net architecture [21]. U-Net, discussed in Subsection 3.1.1, solves the problem of the vanilla ConvLSTM network by contracting path with pooling layers and symmetric expanding path with upsampling layers that are mutually connected in individual levels.

Unitos4 is depicted in Figure 4.2. For a simplicity, the figure shows one input frame and one output frame. However, the real input for the network is a sequence of nine frames. All **ConvLSTM2D** layers have convolutional kernels of size 3×3. All of these layers are of type many-to-many (every time-step returns an output) except the last one of type many-to-one (only last time-step returns an output). The reason for many-to-many is because we want to transfer information of each time-step through the whole network. However, for the output, we are interested only in the prediction from the entire sequence. An input of the first ConvLSTM layer is a sequence of nine frames (patches), each of them of size $64 \times 64 \times 1$. An input of every other ConvLSTM layer is a sequence of nine 3D hidden states.

Conv2D layers apply 2D convolutions through the whole deep of a 3D input structure.

Convolutional kernels with size 3×3 achieve better results than any larger size in numerous models of deep neural networks. As we could see in our experiments with the Primitos3 network, on the one hand, larger kernels are able to capture a larger context, but on the other hand, they are followed by considerable difficulties. In contrast, Unitos4 captures more global context by embedding pooling layers. Therefore, we do not see any reason for larger kernels and all of them, in ConvLSTM2D as well as Conv2D layers, are of size 3×3.

MaxPooling2D decreases the size of the input to half by outputting maximal values from non-overlapping regions 2×2. We have chosen the size to lose as little information as possible in one pooling layer.

UpSampling2D is performed by repeating rows and cols to obtain the output of the double width and height.

Concatenation layer concatenates outputs from two layers, one from contracting path and one from corresponding extracting path. This layer is essential for combining context and localization.

Each Conv2D except last one, each Pooling2D, and each Upsampling2D layer is wrapped into **TimeDistributed layer** that applies the wrapped layer to each time-step.

The number of **filters** of each layer we can see in Figure 4.2.

BatchNormalization [26] is applied after each convolutional layer. It normalizes activations of the previous layer at each training batch with mean close to zero and standard deviation close to one. Batch will be explained more in details in Section 6.2. A consequence of batch-normalization are the activations that are not saturated. In other words, it prevents gradient vanishing. Hence, batch-normalization improves learning as well as accelerates training.

Stride for convolution has been set to 1 in every direction to obtain as much information from an input as possible. Besides, higher stride causes a more significant decrease in input–output dimensions. That is an undesirable consequence since we want an output image as high resolution as possible. Although the stride is of minimal possible value, an output from the convolution with 3×3 kernels is still smaller by 1-pixel border from a convolution input. Applying n convolutions would reduce the image size by n-pixel border. In order to prevent this size reduction, we applied **padding** to the same size in each convolutional layer.

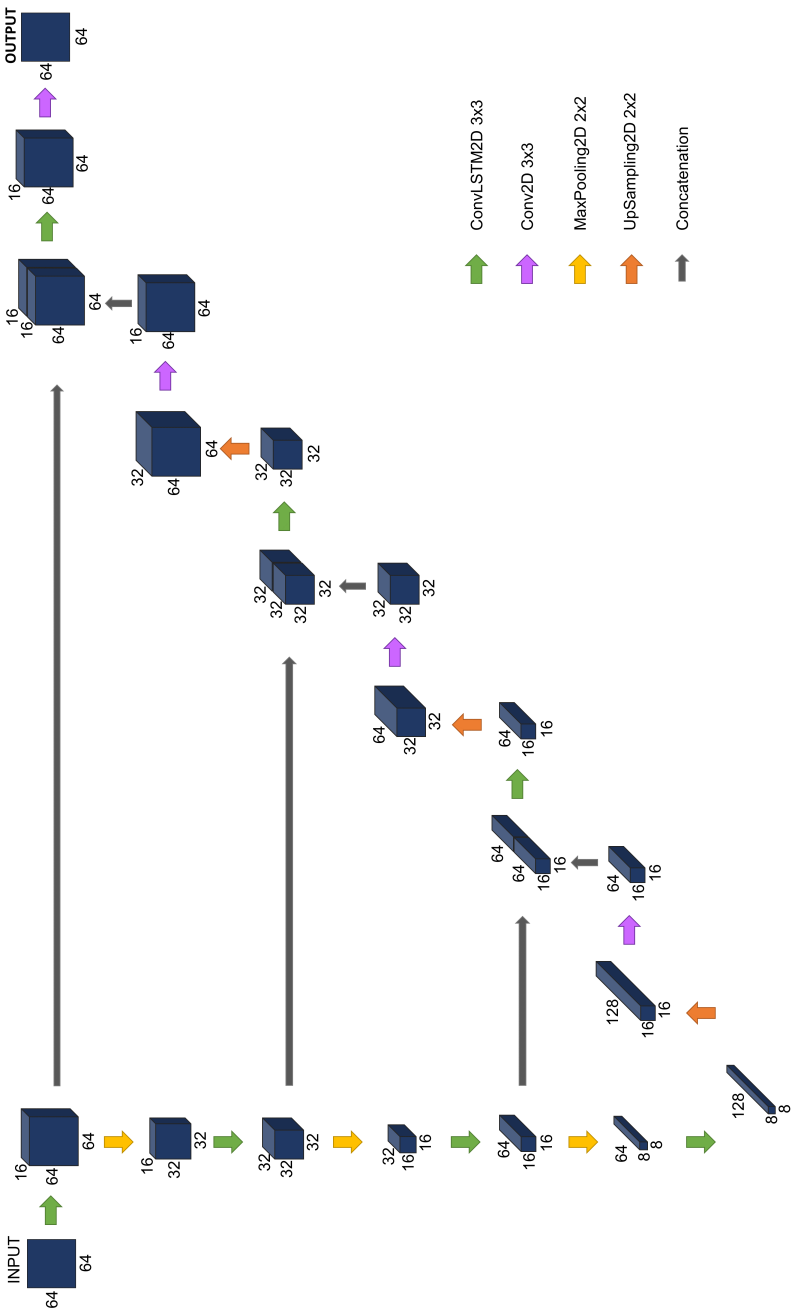
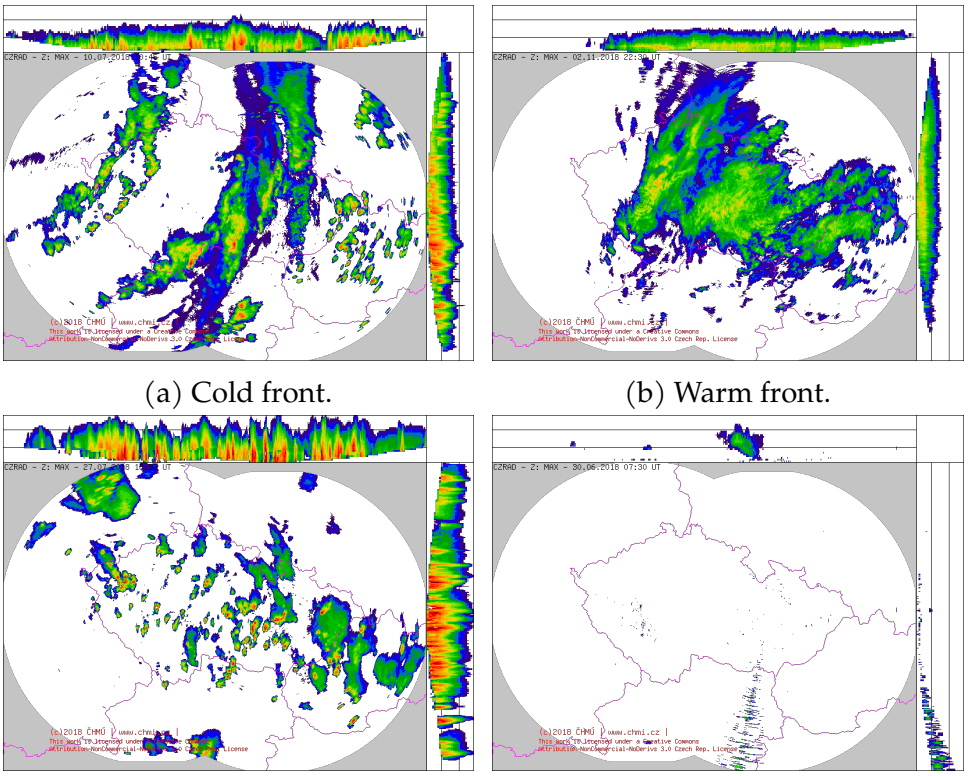


Figure 4.2: **Unitos4**: neural network architecture with U-net structure consisting of ConvLSTM layers.

5 Dataset

The sufficiently representative and large dataset, and appropriate data preprocessing are crucial for a robust neural network. In this chapter, we describe dataset that we have used in the implementation part of this thesis. Later on, we review our data preprocessing steps.

For the implementation part and analyses, we have used weather radar maps provided by the Czech Hydrometeorological Institute (CHMI). CHMI nowcasting web-portal available to the public [1] shows the development of precipitation in the Czech Republic. Examples of such CAPPI images are shown in Figure 5.1



(a) Cold front. (b) Warm front.

(c) Convective showers and storms. (d) Radio interference.

Figure 5.1: Examples of CAPPI images. *Source:* Czech Hydrometeorological Institute [1].

Weather radars measure precipitation reflectivity in dBZ[1]. Possible values are approximately from 4 dBZ up to 60 dBZ. The values are divided into 15 intervals distant from each other by four units. These intervals are mapped to a color scale that is used for visualization of different reflectivities on the map.

The reflectivity is proportional to the rain-rate measured in mm/hour. They are in the exponential relationship and the rain-rate can be computed by Marshall-Palmer formula. Table 5.1 depicts the approximate conversion of reflectivity to rain-rate and color. While rain-rate about 0.1 mm/h represents negligible precipitation that need not fall to the Earth's surface, higher values represent gradually light rain, showers, and storms [1]. The value of roughly 100 mm/h corresponds to hailstones or ice pellets.

The time gap between each two adjacent CAPPI images is 10 minutes. The original dataset used for the network training before any preprocessing contained 14 470 radar images from 23. July 2018 to 1. November 2018. Since not all the days were rainy, we have filtered away images with no rain or hardly any. The filtering method is described in the following section along with other preprocessing steps.

On several images, there are faulty reflections as a consequence of reflections from airplanes, noise and instability of the weather radar, interference of radar measurements with devices radiating a signal of a near frequency (5.6 GHz band [27]), etc. Unfortunately, the consequence of these interfering factors might affect the accuracy of prediction. However, due to a large set of data and problematic filtering of the noise and interference, our dataset contains images with these artifacts. From observing the dataset, we assume

1. dBZ - decibels of Z

Table 5.1: Approximate conversion of rainfall metrics. *Source:* Czech Hydrometeorological Institute [1].

Reflectivity [dBZ]	Rain-rate [mm/h]	Color in CAPPI image
$4-16$	< 0.5	purple, blue
$20-32$	4	green
$36-44$	10	yellow, orange
$48-56$	70	red
60	> 100	white

that the noise does not occur in such a huge amount that it would have a significant negative impact on the predictions.

5.1 Data Preprocessing

Before any preprocessing, the images we have worked with were indexed CAPPI images with 16 possible values: 15 positive values for different rain intensities and zero value represents no rain.

First, we have converted images from indexed to grayscale in order to eliminate the ambiguity of index order and to simplify processing of the images.

Because the network input is a sequence of frames, we created a dataset containing sequences, each sequence containing 10 images: 9 for input and 1 for target output (ground truth). Nine input frames correspond to 90 minutes. All sequences are independent, in other words, one frame cannot occur in two different sequences. We have filtered sequences with no or hardly any rain in order to prevent of overwhelming the network with irrelevant data. The threshold for the filtering was less than 1000 px with a reflectivity less than 16 dBZ, which corresponds to negligible rain intensity of less than 0.5 mm/h.

In the next step, we divided the images into training, validation, and testing datasets. These datasets were assembled in the following way: from each consecutive 10 sequences, we assigned sequences 1..4 into the testing set, sequence 5 into the validation set, sequences 6..9 into the training set again, and the sequence 10 into the testing set. With this approach, each set should be relatively independent and should constitute a representative subset of the whole dataset. In addition, validation and test data should be sufficiently distant to provide a valid evaluation.

The original images have size of 597×377 px. To train a deep neural network on the images of that size would be extremely inefficient (if even possible), considering the performance of current GPU cards. Therefore, we have partitioned the images into patches 90×90 px for our initial model Primitos. The size was chosen to be large enough to capture important information and not to be too large to train a deep network with patches of that size. Moreover, thanks to this step, we increased the number of samples in the dataset.

We have partitioned the images into the patches displaced by 30 px from each other in both directions. This scheme offers $2/3$ overlap. Because the original image size 597×377 px is not divisible by 30, we have omitted borders. The final size of the images that were partitioned into the patches was 570×360 px.

After we encountered the deficiencies of Primitos, we cropped the patches and used them for Unitos as well. Pooling layers of Unitos reduce in-

put size by factor 2 in both spatial dimensions. Subsequently, upsampling layers double the width and height until they reach the input size. Unitos4 performs both subsampling and upsampling three times. In order to avoid inconsistencies in the dimensions in individual layers, we need spatial input dimensions equal to power of two, e.g., 32×32, 64×64, or 128×128. Hence, we cropped the patches to the size 64×64 px.

After the processing of patches, several of them were blank or almost blank as the rain normally does not cover the whole area of the Czech Republic. We have removed sequences that contained at least one patch with less than 300 px with a value less than 16 dBZ.

Examples of such patches that were used for network training and prediction are shown in Figure 5.2.

To enlarge the training dataset even further, we have obtained additional sequences by "dense sampling" of adjacent independent sequences. Let the $t_1, \ldots, t_{10}, t_{11}, \ldots, t_{20}$ be the frames from adjacent training sequences. Then, we can generate 10 different interleaved sequences $s_i = \{t_i, \ldots, t_{i+10}\}$. Since we have typically 4 training sequences in a row, we can generate up to 40 sequences from the 4 independent ones.

After all the mentioned preprocessing steps were completed, the training dataset contained in total 148 220 samples, the validation dataset 2 573 samples, and the test dataset 2 193 samples.

Appropriate data normalization is crucial for network learning. If the data is from an inappropriate data domain, the network is not able to learn anything relevant. This is due to the small gradient magnitude of the boundary values in activation functions, such as sigmoid and tanh. They are displayed with their derivatives in Figure 3.6. In the ConvLSTM cell, both sigmoid and tanh are used. Taking into account the properties of sigmoid and tanh, together with the properties of their derivatives and backpropagation process, we tested several reasonable possibilities for data normalization. Finally, normalization to $\langle 0.25, 0.75 \rangle$ has achieved the best result.

Target data (ground truth) used for network training is normalized as well as input data.

Predicted output values are only approximately in the range $\langle 0.25, 0.75 \rangle$. Some of the predicted values may exceed this range. Therefore, the postprocessing of predicted images includes inverse steps to normalization followed by clipping of exceeded values.

Comparison of our results with the method currently used by the Czech Hydrometeorological Institute is given in Section 7.4. The CHMI prediction images were processed in the same way as the testing dataset.

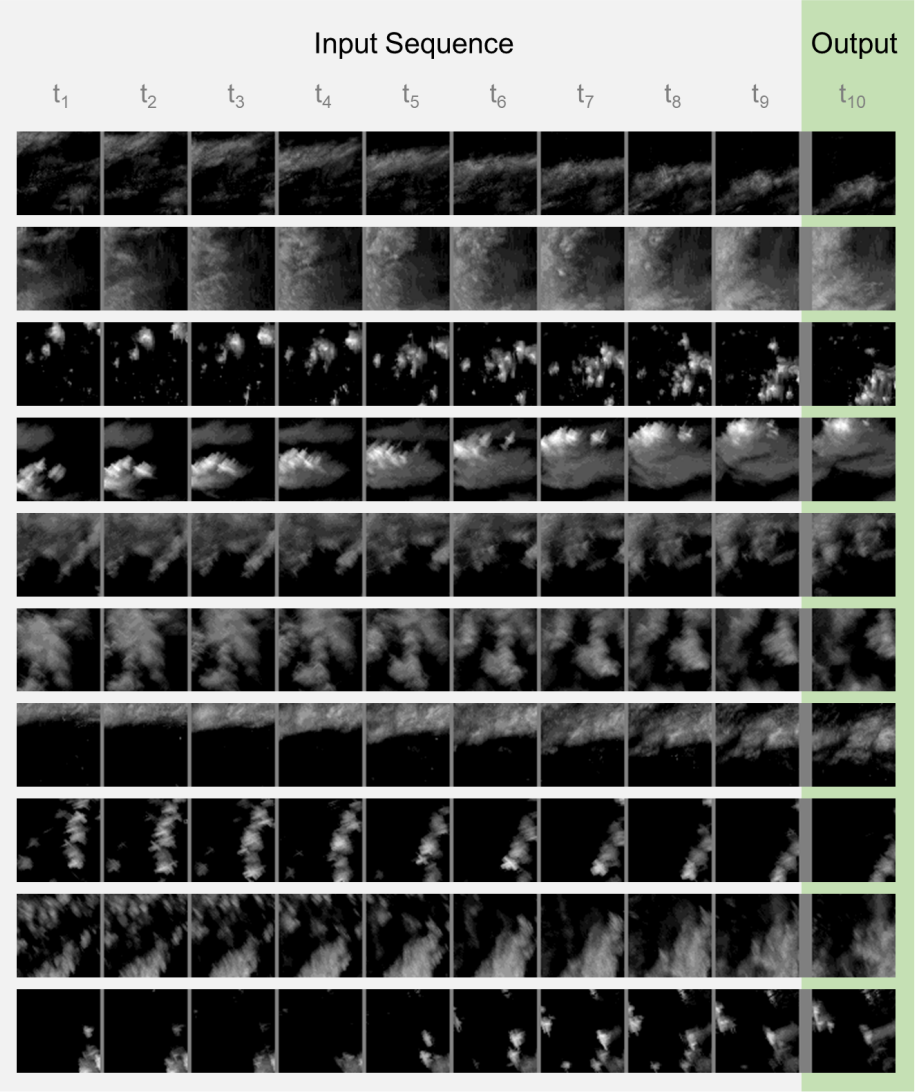


Figure 5.2: **Samples of patches** from several weather radar images showing various types of precipitation. **Input sequence** consists of 9 patches capturing the development of rain in 90 minutes from time t_1 to t_9. It is used as an input for network training/prediction. **Output** column shows the real weather at time t_{10}, which is used as an output for network training or as ground truth for evaluation of network prediction. *Source:* The patches are cropped and converted to gray-scale from CAPPI images from the CHMI [1].

6 Implementation

In this chapter, we summarize the tools we used for the development of our network in Section 6.1. Later, in Section 6.2 we discuss the training process with its parameters such as optimization algorithm, learning rate, and a loss function.

6.1 Software

We preprocessed data for training, evaluate our final predictions and compare them with the currently used method in MATLAB because of its considerable support of image processing

We have written all source code for development of network architecture, training, testing, predictions, and visualizations in Python using Keras library [24] with TensorFlow backend [28]. Keras is a high-level library for deep learning, written in Python, supporting convolutional networks and recurrent networks, as well as ConvLSTM layers we used in both of our architectures. TensorFlow is an open source machine learning framework developed by Google. It has strong support for deep learning and, in addition, provides TensorBoard extension, a tool for visualization of network learning.

TensorFlow requires NVIDIA GPU drivers, CUDA Toolkit [29] for parallel computations on GPU, and cuDNN [30] library for GPU acceleration of deep neural networks.

For management of Python packages, we used open source Anaconda platform [31]. It comes with a large number of data science packages, in addition to Conda, a package management system that is capable of conveniently managing dependencies and virtual environments. In addition to the default packages that come with Anaconda 4.5.4, the most recent version at the time of development, we have installed TensorFlow, TensorBoard, and Keras along with their dependencies. Furthermore, the h5py [32] package was updated to version 2.8.0.

Table 6.1 summarizes all the mentioned tools we used for the development of our network along with the versions.

Table 6.1: Tools with used in the implementation

Tool	Version	Description
CUDA Toolkit	9.0	Parallel computing platform for GPU accelaration
cuDNN	7.0	GPU-accelerated library for deep neural networks
NVIDIA GPU drivers	390.30	GPU drivers
Anaconda	4.5.4	Platform for Python packages and environments management
Python	3.6.6	Programming language
keras-gpu	2.2.4	Deep learning library
tensorflow-gpu	1.7.0	Machine learning library (Keras backend)
h5py	2.8.0	Library: a Pythonic interface to the HDF5 binary data format

6.2 Training

The training of our proposed network Unitos4 was carried out on a single GPU GeForce GTX 1080 with 8 GB memory. The final model was trained in 11 epochs divided into two parts due to manually adjusting the learning rate, explained later in this chapter. The training took approximately 19 hours with 98 minutes for one epoch.

Learning of neural network involves optimization that minimizes loss function. There are two types of the optimization algorithm for neural networks. First-order optimization algorithms are based on first derivatives. Well-known is Gradient Descent that minimizes loss (dependent variable) by moving weights (independent variables) in small steps in the opposite direction of the derivatives. Second-order optimization algorithms are based on second derivatives, for instance, Newton's method using a second-order Taylor series [20].

The optimization algorithm has three variants. The first variant is batch optimization algorithm that processes the whole dataset at once for one update of training parameters. The second is stochastic, which update the parameters after processing of each sample. And the third one is mini-batch that processes just a subset of the whole dataset at a time for one update.

The mini-batches are often generalized and called also batches. Larger mini-batches produce a more accurate estimate of the gradient but the size is limited by GPU memory. It is common to use batch size power of two because of GPU optimization [20].

Gradient Descent optimization algorithms have several implementations, such as well-know RMSProp[1], Adadelta [34], Adam [35] and its variants, and others [19]. We have tested these optimizers in our preliminary experiments with Primitos, and based on our observations with convergence and stability, we have decided to use RMSProp. Moreover, RMSProp is also recommended for recurrent networks on Keras websites [24]. All RMSProp parameters were left to the default values except the learning rate.

The **learning rate** was set before training to a default value of 0.001. As we observed higher instability after the first epoch of training, we have reduced the learning rate to 0.0001 and continued with training. After ten epochs of the second iteration of training, the validation loss started to increase slightly while training loss was still slowly decreasing. This behavior is typically a sign of overfitting since network predictions are still improving on the dataset that is seen by the network but not anymore on the set that the network does not see. The training process is displayed in Figure 6.1.

The total number of samples used for training was 148 220. Since the samples are from similar data distribution, as they are all weather radar images, we trained the network with **batches** of size only four.

We used pixel-wise Mean Squared Error (MSE) **loss function** for training. It is computed using the formula

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

where Y is ground truth and $\hat{Y}$ is prediction. The loss is computed on every sample and then averaged or summed over the batch. Subsequently, training weights are updated accordingly.

We used the MSE loss because we want to approach predicted intensity as close as possible to the ground truth, namely at every pixel. The loss in our final Unitos4 model is computed on predicted images and target images (ground truth) cropped by 1-pixel border due to the convolution and padding explained in Section 4.3.

Because the dataset was normalized to the interval $\langle 0.25, 0.75\rangle$, possible loss values are from the interval $\langle 0, 0.25\rangle$. Identical images would have zero

1. RMSProp algorithm is unpublished, first proposed by Geoff Hinton [33]

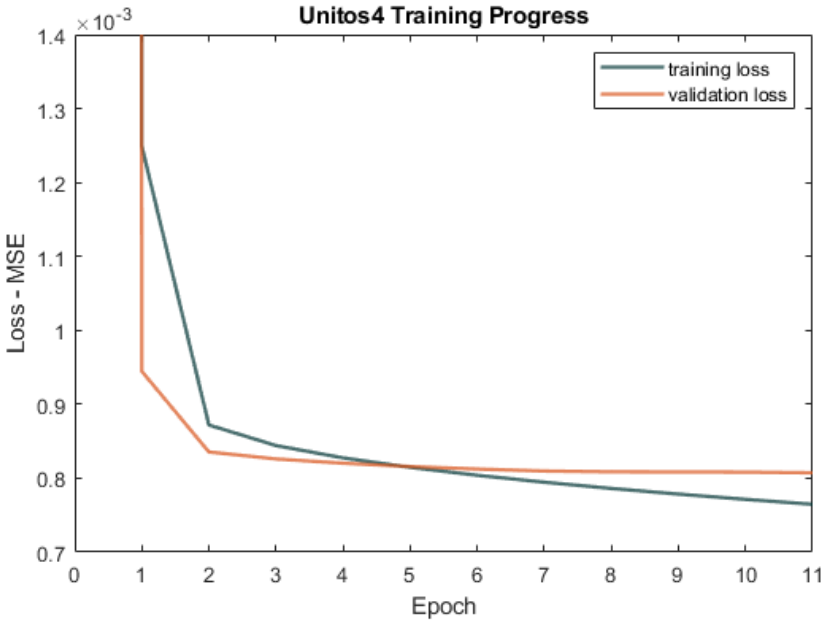


Figure 6.1: Unitos4 training progress visualizing reduction of Mean Squared Error for training and validation dataset.

loss, while maximally different images (completely white vs. completely black) would have the loss value 0.25.

7 Evaluation and Results

In order to evaluate and demonstrate the performance of our method, we performed several experiments and evaluations. First, we analyze the properties with respect to the Ground Truth, and then, we also compare our results with the baseline and with the existing CHMI prediction algorithm. We also analyzed the properties of the network in order to verify, how the network works, and what it has really learned inside.

All evaluations were computed for our final Unitos4 model on the test dataset that had been separated from training and validation data at the beginning of our research and we did not use it for any training or tuning of any network. We computed all results and comparisons in this chapter after we had defined the final model.

We have divided the problem of precipitation nowcasting into two main subproblems in Section 4.1. The first is an accurate prediction of rain cells movement and the second is an accurate prediction of their intensity development.

We use a different metric for evaluation of each subproblem. For evaluation of movement, we use the Jaccard coefficient, and for the evaluation including also accurate prediction of the intensity values, we use Mean Square Error.

Predictions of weather radar images by our network Unitos4 are visualized in Figure 7.1. There are shown several samples of various types of precipitation. Unitos4 network was trained on patches that are displayed in strips as time sequences in the range $\langle t_1, t_9 \rangle$. Further, there is shown ground truth (GT) at time t_{10} along with our prediction for t_{10} computed from the input sequence. In addition, our prediction is compared to GT.

We can observe from these samples that Unitos4 network is able to predict the velocity of movement of rain cells in the right direction. Moreover, it quite accurately predicts the intensity development of precipitation. There are slight details that Unitos4 does not predict accurately on the pixel level, in particular, the "noisy" structure of precipitation on CAPPI images. Unitos4 has a tendency to form slightly more homogeneous regions. This may be caused by the structure of the architecture. However, it should be reminded that there is a certain inaccuracy and noise in weather radar measurements, which can have a negative impact on the accuracy of the forecasting.

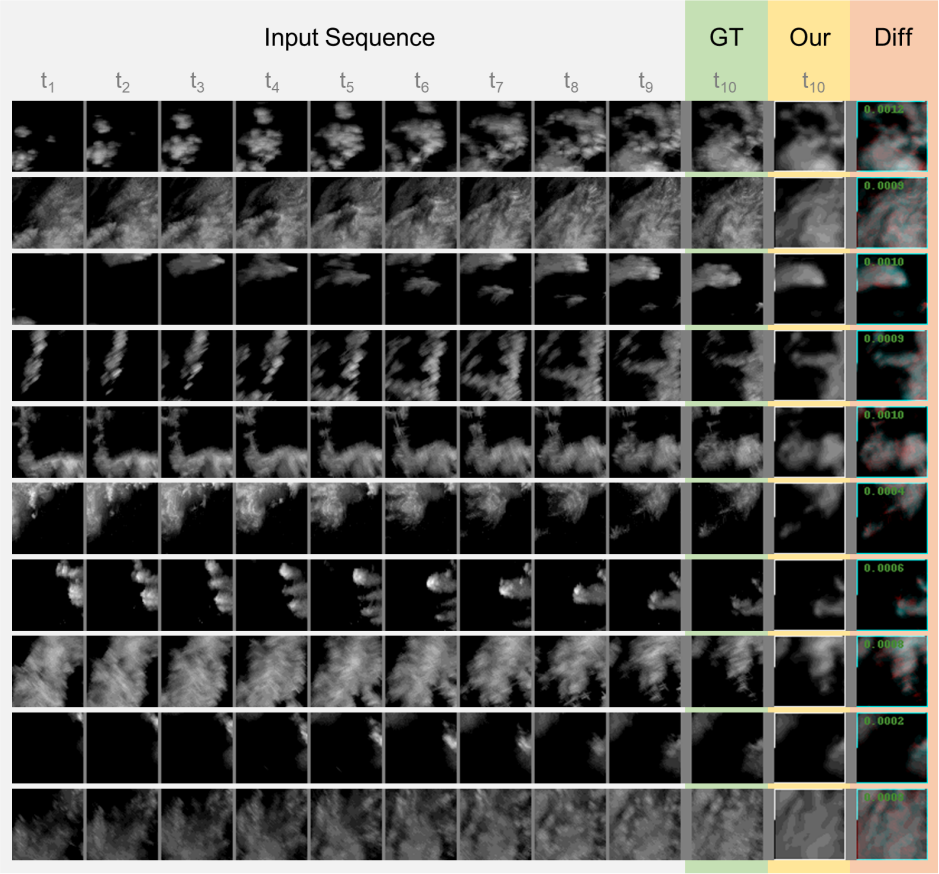


Figure 7.1: **Forecast by our Unitos4 network.** The samples of patches from several weather radar images depict various types of precipitations. **Input sequence** consists of 9 input patches capturing the development of rain in 90 minutes $(t_1, \ldots, t_9)$. Adjacent patches (t_i, t_{i+1}) are 10 minutes apart. **GT** (ground truth) shows the real weather at t_{10}. **Our** comlumn presents predicted frames from the input sequence for the time t_{10}. **Diff** column visualizes the difference between GT and the prediction. The red color represents the regions that are underexposed and the cyan color represents the regions that are overexposed in our prediction with respect to GT. The number is Mean Squared Error between GT and our prediction. The 1-pixel border of patches was not taken into account in Unitos4 training, as explained in Section 6.2. *Source:* Input sequences and GT are from the CHMI (modification: cropped, gray-scale) [1].

7.1 Evaluation of Movement Prediction

To evaluate the ability of movement prediction for the rain cells, i.e., the direction and velocity, we have first obtained the binary images by thresholding the intensities, where foreground represents areas with "enough" rain.

In order to compare binary masks, we used the Jaccard coefficient, also known as the Jaccard index, that measures the similarity between two sets as the ratio between the size of their intersection and the size of their union. It is computed using the equation:

$$\text{jaccard}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

The Jaccard coefficient is always in the interval $\langle 0, 1 \rangle$ where 0 is for the disjunctive sets, while 1 is for identical sets.

The images with an extremely small area (amount of pixels) with the sufficient intensity typically consists of noise only. For that reason, we have excluded samples with less than 16 px of foreground in the expected image. After the filtering, the total number of 2058 samples were used for the computation of statistics.

The last thing to decide is, which intensity level represents "enough" rain. If we use intensity 1, we will consider also "blue" regions, which typically do not represent rain, but only the high concentration of humidity, or the rain, that do not fall all the way to the ground, but dissipates somewhere in the middle of its trajectory. On the other hand, the intensity level 4 typically represents light rain (drizzle), that really fall onto the ground. Therefore, we present results for both mentioned intensity thresholds 1 and 4.

Both distributions of Jaccard coefficients are depicted in the Figure 7.2. We achieved the median value of 0.72 and 0.84 for intensity threshold 4 and 1 respectively. The Jaccard coefficient for lower threshold is higher due to the fact, that areas with low intensity are larger, therefore the overlap for the particular movement direction and speed is higher than for smaller areas with higher intensity.

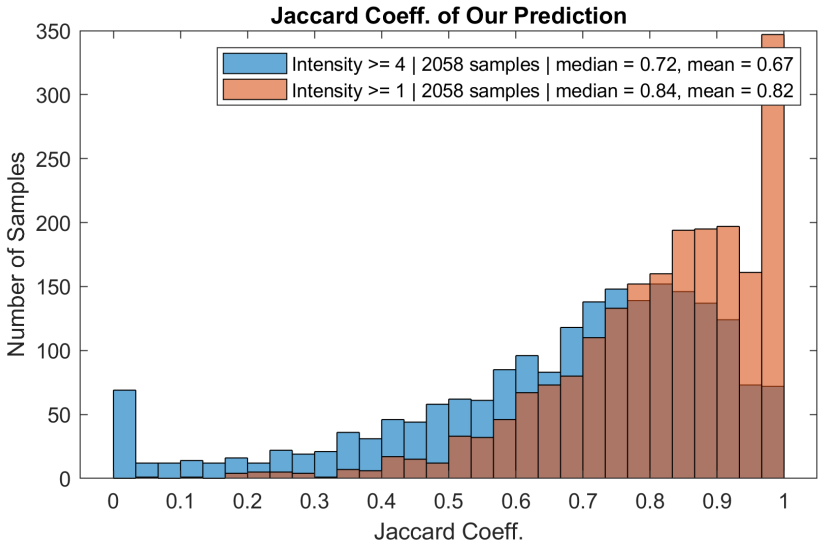


Figure 7.2: Histogram of Jaccard Coefficients for Our Prediction compared with Ground Truth. The blue histogram is for intensity threshold 4, while the red one is for intensity threshold 1.

7.2 Evaluation of Pixel-Wise Precision

To evaluate the intensity development of rain we used Mean Square Error (MSE), the same function that has been used as the loss function for training, discussed in Section 6.2. MSE is able to capture pixel-wise distance of two images on the intensity level. Therefore, we use this metric to evaluate the ability to accurately predict intensity development simultaneously with movement. Since we scaled intensities of predicted images back to the discrete interval $\langle 0, 15 \rangle$, results of MSE are from interval $\langle 0, 225 \rangle$. For recapitulation, the MSE is defined as

$$\mathrm{MSE}(A, B) = \frac{1}{n} \sum_{i=1}^{n} (A_i - B_i)^2$$

We computed MSE statistics from all 2193 samples and we achieved median MSE value of 0.64. The distribution of MSE is shown in Figure 7.3.

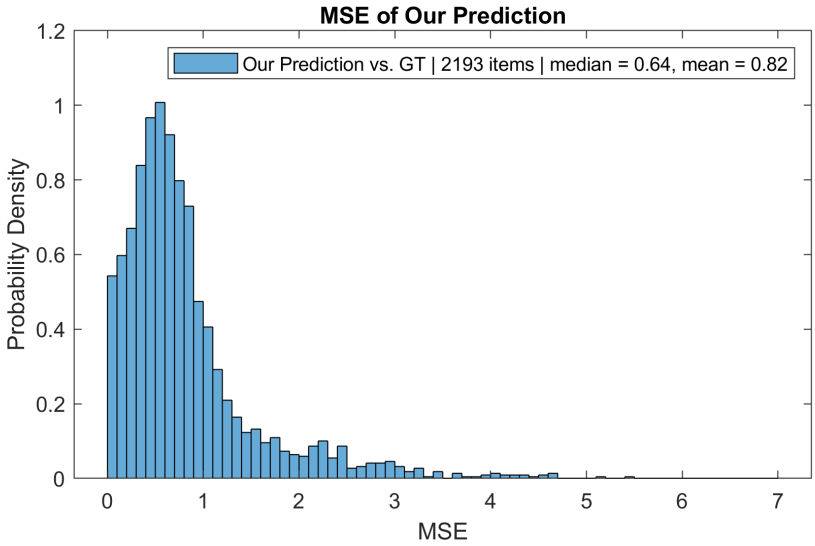


Figure 7.3: Histogram of MSE for Our Prediction compared with Ground Truth.

7.3 Comparison with Baseline

We want to compare the results of Unitos4 network with a baseline. For this reason, we define the baseline as the result of *NULL-predictor*, that essentially copies the last real frame from the input sequence to the output. For the NULL-predictor, we compute the same metrics of the Jaccard coefficient and MSE as were already described above.

To visualize the difference, we made the scatter plot as depicted in Figure 7.4. In this plot, each sample is projected into a 2D space where the horizontal axis represents the Jaccard coefficient for NULL-prediction, while the vertical axis represents the Jaccard coefficient for our prediction.

Points on the identity line represent samples with Jaccard coefficient of our prediction equals to Jaccard coefficient of the last frame. Samples with a better Jaccard coefficient for our prediction than the last frame are projected above the identity line because the coordinate of the sample on the vertical axis is higher than on the horizontal axis. Samples laying below the line represent the opposite case.

To summarize, we can see that significantly more samples are projected above the identity line. All these samples achieve the Jaccard Coefficient above the baseline. In other words, our Unitos4 network is able to predict movement better than the NULL-predictor.

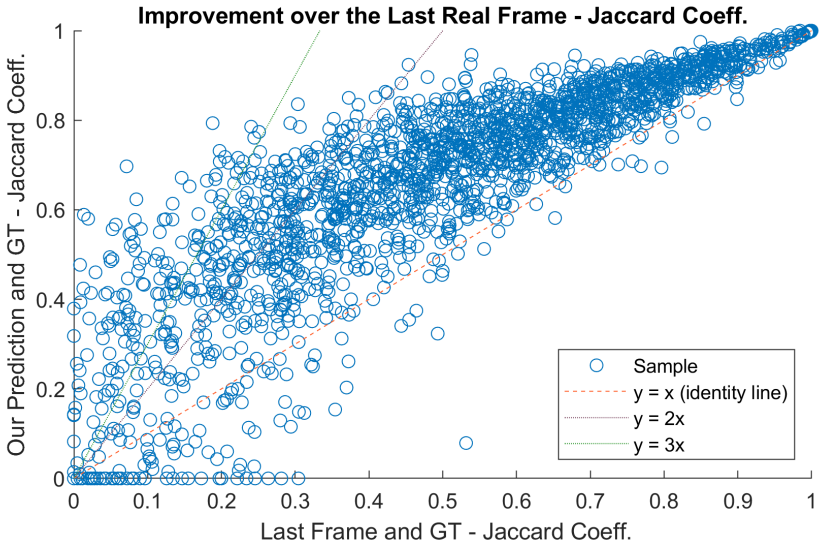


Figure 7.4: Comparison of the Jaccard coefficient for our prediction and NULL-prediction (the last real frame). The Jaccard Coefficient is computed for both predictions with ground truth (GT).

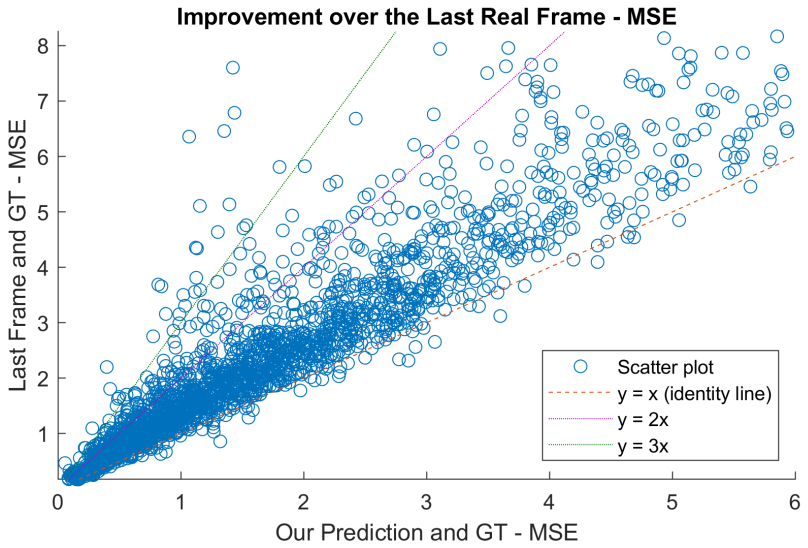


Figure 7.5: Comparison of Mean Square Error (MSE) for our prediction and NULL-prediction (the last real frame). MSE is computed for both predictions with ground truth (GT).

In order to evaluate the intesity development of our prediction in comparison with NULL-predictor, we show MSE in Figure 7.5. It should be noted that a lower MSE between two sets means that the sets are more similar compared to the Jaccard coefficient, where a higher result corresponds to a higher similarity of the sets. Therefore, the axes of the MSE plot are reversed in contrast to the previous Jaccard plot. With the reverse axes, samples for which Unitos4 predictions are closer to GT than last real frame are again displayed above the identity line.

We can see that for MSE as well as for the Jaccard there are significantly more samples above the identity line. This result can be seen as the ability of Unitos4 network to predict next frame better than NULL-predictor.

In addition, we depict the evaluation of MSE in the histogram plot in Figure 7.6. It can be easily seen, that the distribution of our MSE is shifted to the lower values, that again shows the better prediction of Unitos4 against the baseline.

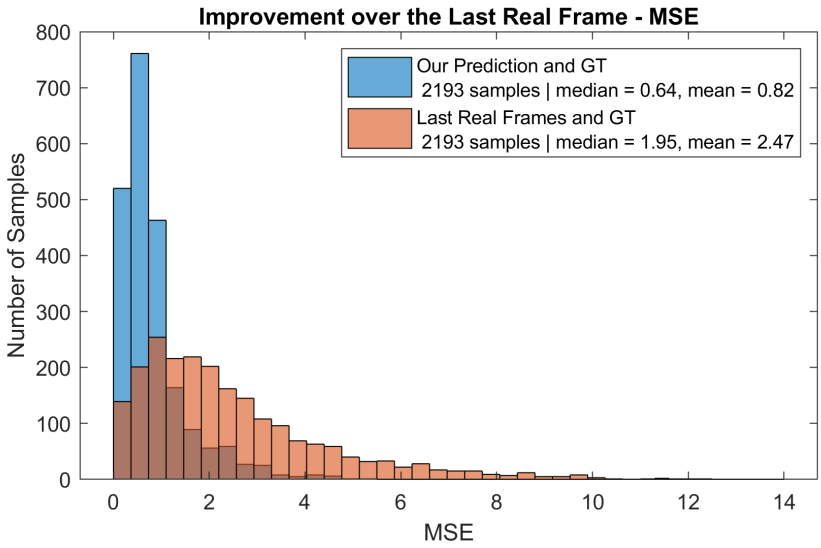


Figure 7.6: Comparison of Mean Square Error (MSE) for our prediction and NULL-prediction (the last real frame). MSE is computed for both predictions with ground truth (GT).

7.4 Comparison with CHMI Prediction

In this section, we compare our prediction produced by Unitos4 with the currently deployed method for nowcasting by the Czech Hydrometeorological Institute (CHMI) [1]. Since we were missing 459 CAPPI images forecasted by CHMI at times corresponding to our test samples, we have excluded all the test samples that were missing CHMI counterparts. Consequently, comparison of our prediction with CHMI prediction was evaluated on 1734 samples in contrast to evaluation in Section 7.3 computed on 2193 samples.

Representation of results depicted in scatter plot and in histogram plot has been explained in the previous sections.

In Figure 7.7, we can see a comparison of our prediction with CHMI prediction by plotting the Jaccard coefficients in the scattered plot. It can be observed that significantly more samples are above the identity line. This corresponds to the fact, that our predicted results are more similar to GT than the predictions produced by CHMI. In other words, Unitos4 model can predict movement better than the CHMI prediction algorithm. The same conclusion can be derived from the histogram plot shown in Figure 7.8 where the distribution of the Jaccard coefficient for our prediction is closer to ideal prediction (Jaccard ceofficient=1) than CHMI prediction.

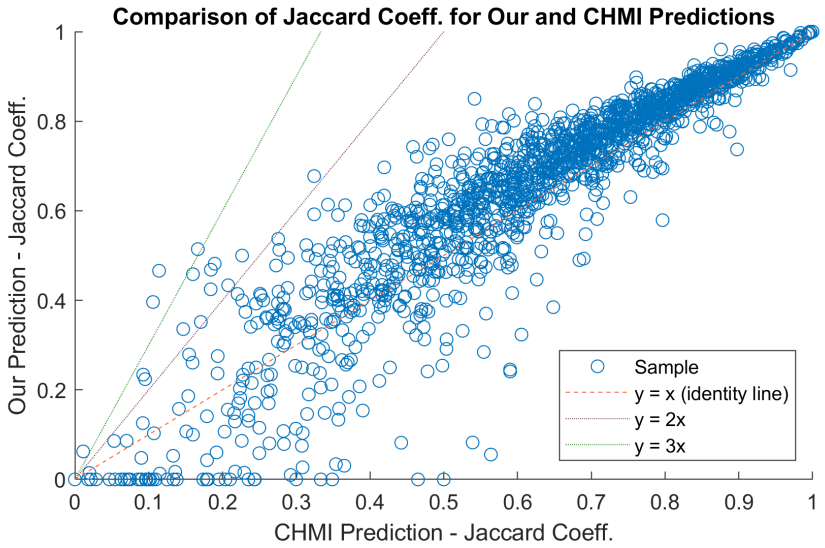


Figure 7.7: Comparison of the Jaccard Coefficient for our prediction and CHMI prediction. The Jaccard Coefficient is computed for both predictions with ground truth (GT).

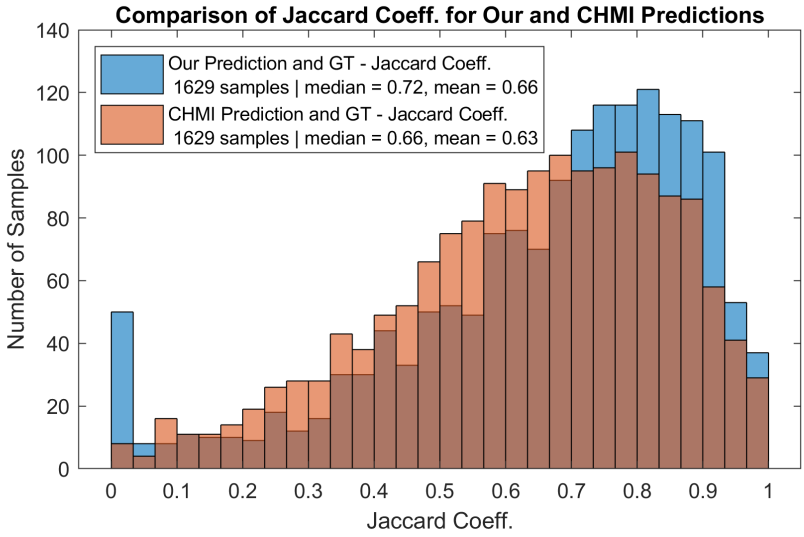


Figure 7.8: The histogram of the Jaccard Coefficients for our prediction and CHMI prediction. The Jaccard Coefficients are computed for both predictions with ground truth (GT).

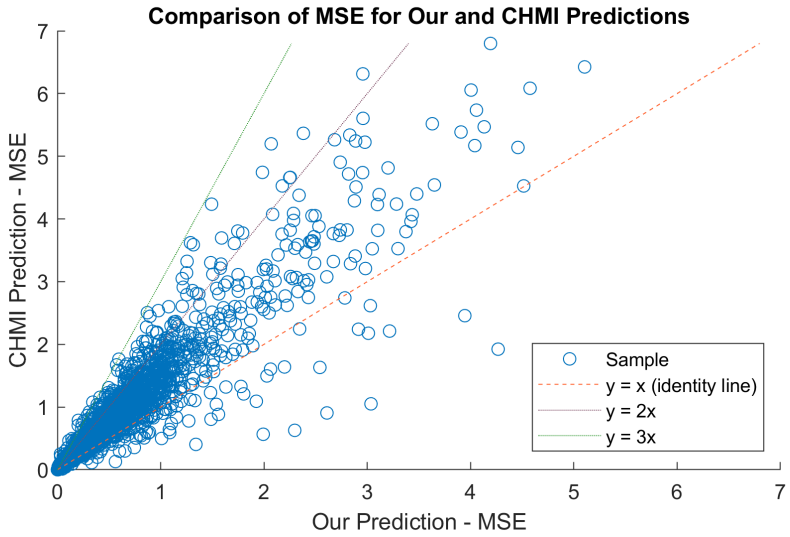


Figure 7.9: Comparison of Mean Square Error (MSE) for our prediction and CHMI prediction. MSE is computed for both predictions with ground truth (GT).

Comparison of MSE for our prediction and CHMI prediction is depicted in Figure 7.9. It is apparent that considerable more samples are located above the identity line that corresponds to higher MSE for CHMI prediction than our prediction. This can be conceived as our prediction able to predict movement along with intensity of rain better than CHMI. The same comparison is shown in Figure 7.10 visualizing our prediction with distribution closer to ideal prediction than CHMI prediction.

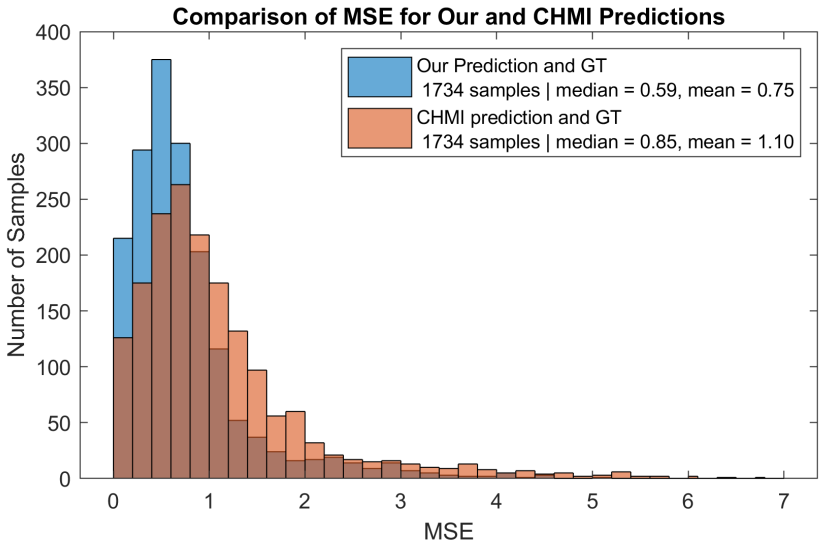


Figure 7.10: Comparison of Mean Square Error (MSE) for our prediction and CHMI prediction. MSE is computed for both predictions with ground truth (GT).

7.5 Comparison of Full-Frame Merged Images

The original radar images, we get from CHMI, are of size 597×377 px. As we mentioned beforehand, we generated smaller patches from each of the image. After the neural network produces the prediction for each individual patch, we want to merge them together to obtain the large predicted result, which we call *full-frame image*.

Thanks to the chosen patching scheme, most of the pixels in the full-frame image are covered by four patches. During the merge, we average all possible values from individual patches to eliminate artifacts on the patch boundaries. After the merging and averaging, the values are leveled and projected back from the neural-network range $\langle 0.25, 0.75 \rangle$ to the indexed values $\langle 0, 15 \rangle$.

Four examples of such merged images are shown in Figure 7.11 and Figure 7.12, while more examples are shown in Appendix A. All figures demonstrate our predicted result, ground truth and also the CHMI prediction for comparison.

We generated a preview of 12 full-frames images. For these images, we also computed the MSE and Jaccard coefficient (for both threshold levels 1 and 4), and we compared our results with CHMI prediction. The statistical values from these 12 full-frames images are shown in Table 7.1 and Table 7.2. And though it is not an exhaustive set of images, the results show, that our method is able to achieve better values for all statistics.

Table 7.1: Full-frame evaluation of Jaccard coefficient on 12 images.

	Jaccard (int.≥ 4)			Jaccard (int.> 0)		
	median	mean	std. dev.	median	mean	std. dev.
Our	**0.7373**	**0.7813**	**0.1372**	**0.8234**	**0.8128**	**0.0706**
CHMI	0.6888	0.7215	0.1401	0.8150	0.7896	0.0747

Table 7.2: Full-frame evaluation of MSE on 12 images.

	MSE		
	median	mean	std. dev.
Our	**0.4273**	**0.5902**	**0.4152**
CHMI	0.6879	0.8590	0.4742

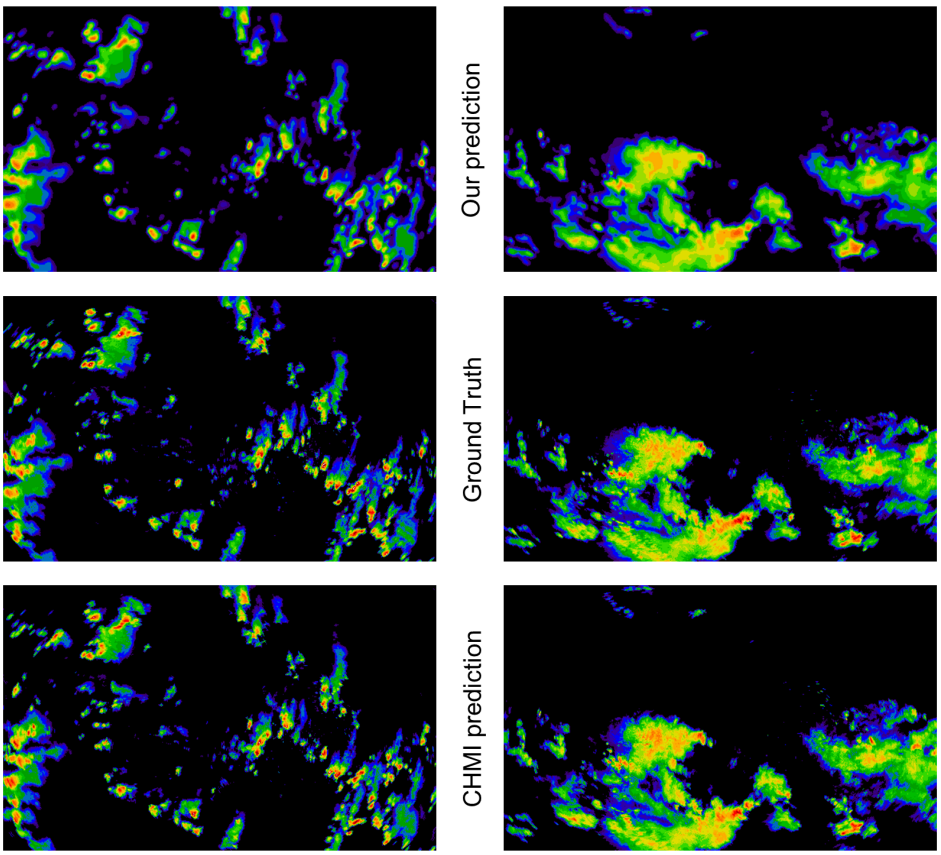


Figure 7.11: Prediction by Unitos4 network compared with ground truth and CHMI prediction. *Source:* GT and CHMI prediction are from the Czech Hydrometeorological Institute [1].

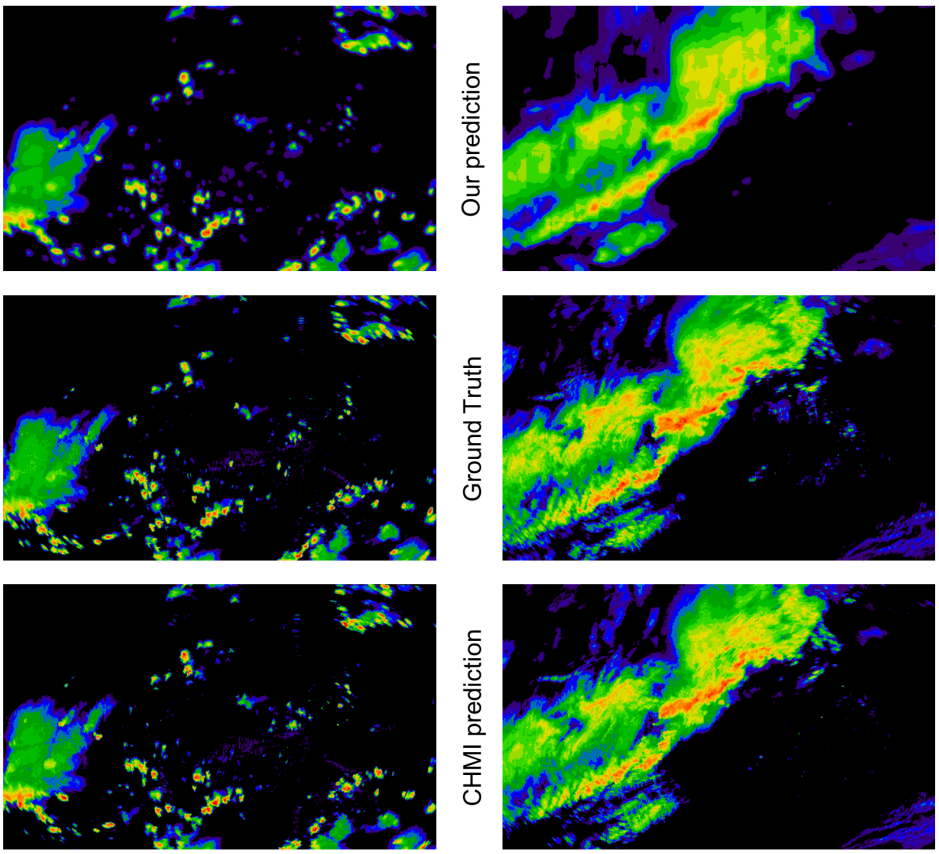


Figure 7.12: Prediction by Unitos4 network compared with ground truth and CHMI prediction. *Source:* GT and CHMI prediction are from the Czech Hydrometeorological Institute [1].

7.6 Understanding of the Prediction Process

We also conducted an experiment to discover how significant is the impact of the individual time-steps to the prediction.

Figure 7.13 visualizes an input sequence of nine frames from time t_1 to time t_9 and ground truth frame at t_{10}. Our prediction for time t_{10} is computed from the input sequence and compared with ground truth. We can see that the first sequence (completely empty) correctly leads to an empty future frame. From the other sequences is obvious that the last input frame at the time t_9 has an immense impact on the prediction. In the 9^{th} row, where all the input frames are rainy except the last one, the result is also empty due to the strong impact of the last frame.

Figure 7.14 demonstrates how distant past of rain development is necessary to accurately predict the future frame. For the depicted example, only four last frames were essential for the best possible forecast. Adding frames from the previous time-steps did not improve the prediction in any significant way. On the other hand, these four frames had a considerable impact on the accurate prediction of movement as well as sharpness of the rain cell.

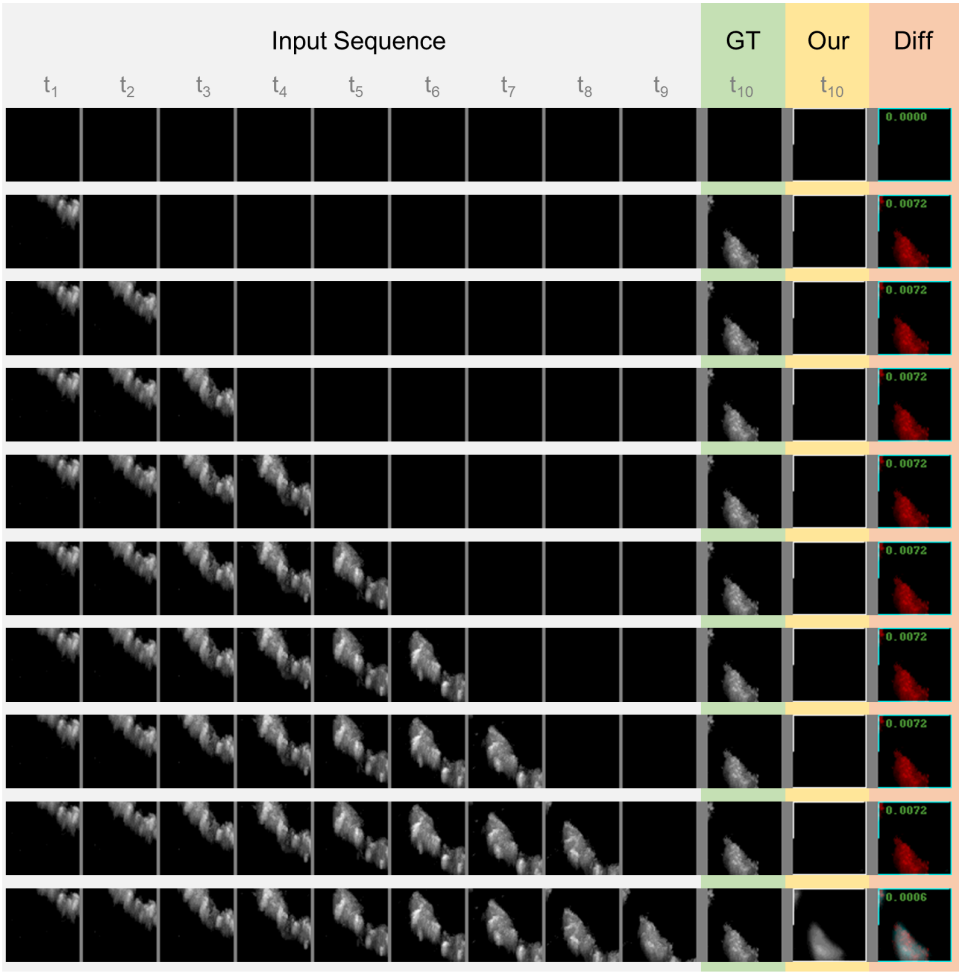


Figure 7.13: **An experiment demonstrating the impact of individual input frames on the prediction of future frame.** Samples consist of patches from CAPPI images visualizing development of rain. *From up to down:* from the empty sequence through gradually appending next frame to the full sequence. *From left to right:* 9 input frames from time t_1 to t_9, **GT**: ground truth at t_{10}, **Our**: our prediction at t_{10} from the input sequence, and **Diff**: difference between GT and our prediction. The red color represents the regions that are underexposed and the cyan color represents the regions that are overexposed in our prediction with respect to GT. The number is Mean Squared Error of our prediction. *Source:* The input sequences and the ground truth frames are cropped and converted to gray-scale from CAPPI images from the Czech Hydrometeorological Institute [1].

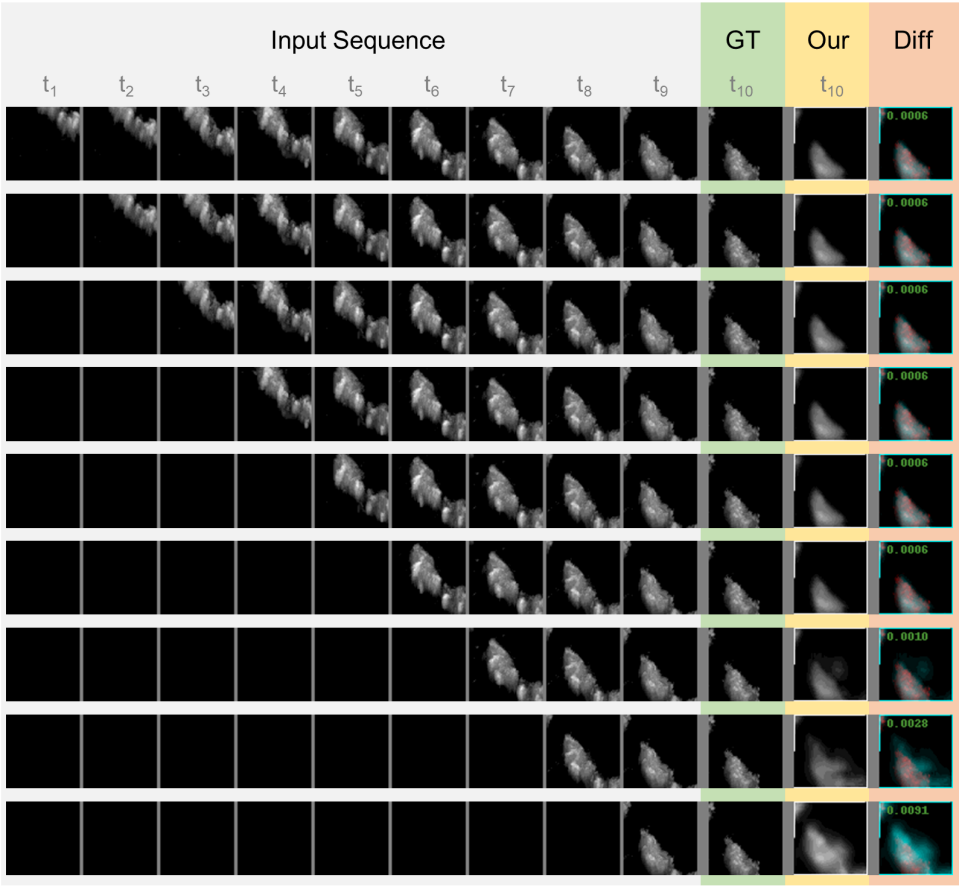


Figure 7.14: **An experiment demonstrating the impact of individual frames of an input sequence on the prediction of the future frame.** Samples consist of patches from weather radar images visualizing the development of rain. *From up to down:* from the full sequence through gradually removing the frames to the sequence containing only last frame at time t_9. *From left to right:* a sequence of 9 input frames from time t_1 to t_9, **GT**: ground truth at t_{10}, **Our**: our prediction at t_{10} from the input sequence, and **Diff**: difference between GT and our prediction. The red color represents the regions that are underexposed and the cyan color represents the regions that are overexposed in our prediction with respect to GT. The number is Mean Squared Error of our prediction. *Source:* The input sequences and the ground truth frames are cropped and converted to gray-scale from CAPPI images from the Czech Hydrometeorological Institute [1].

8 Conclusion and Future Work

In this thesis, we analyzed the problem of precipitation nowcasting, accurate prediction of rain development into near future from weather radar maps. We provided a brief overview of related work and currently used approaches for the problem. Then we reviewed neural network architectures that could be deployed for the forecasting.

We proposed Unitos4 neural network architecture that is formed from ConvLSTM [9] layers into a structure similar to U-Net model [21] consisting of contracting and expanding path. Unitos4 architecture combines the advantages of both models. ConvLSTM layers provide the ability to capture dependencies between time-frames. However, these layers have deficiencies in learning of global context that is essential for network learning a higher velocity. The structure of U-Net is able to extract global context better by pooling layers of contracting path. In addition, it preserves resolution and allows accurate localization by upsampling performed in expanding path.

We evaluated our results by several methods against baseline as well as the currently used method for nowcasting by the Czech Hydrometeorological Institute [1]. Ultimately, prediction by Unitos4 achieved higher accuracy on our test dataset in the all conducted evaluations.

From the presented results, we can see, that our method predicts quite accurately rain movement as well as intensity development. Although, there is a slight difference between our prediction and real measurements at the pixel level. Unitos4 have a tendency to form more homogeneous regions in contrast to real radar images with "noisy" structure. This may be caused by the structure of Unitos4. However, it should be also reminded that the error in prediction may have been already introduced during the training process because of the noisy or erroneous data.

For the future work, we plan to forecast a more distant future than 10 minutes. In addition, we will implement a weighted loss function that may improve network training as well as increase prediction accuracy. Furthermore, we will investigate optimization techniques, such as Dropout [36], to reduce network overfitting. For even higher accuracy, we plan to enhance Unitos4 with residual connections [37].

Besides, we will enlarge the dataset with recently gathered images as well as with data augmentation. In addition, we plan to visualize network feature maps along with training weights in individual layers to get a better understanding of the network learning.

Bibliography

1. *ČHMÚ nowcasting webportal* [online]. Czech Hydrometeorological Institute [visited on 2018-11-18]. Available from: `http://portal.chmi.cz/files/portal/docs/meteo/rad/inca-cz/short.html`.
2. *Aladin* [online]. Czech Hydrometeorological Institute [visited on 2018-12-10]. Available from: `http://portal.chmi.cz/files/portal/docs/meteo/ov/aladin/results/ala.html`.
3. *Medard* [online]. Ústav informatiky Akademie věd České republiky [visited on 2018-12-10]. Available from: `http://www.medard-online.cz/projectInfo`.
4. *The Weather Research and Forecasting Model* [online]. The National Center for Atmospheric Research [visited on 2018-12-10]. Available from: `https://www.mmm.ucar.edu/weather-research-and-forecasting-model`.
5. *Nowcasting* [online]. World Meteorological Organization [visited on 2018-11-11]. Available from: `http://www.wmo.int/pages/prog/amp/pwsp/Nowcasting.htm`.
6. KIMURA, Ryuji. Numerical weather prediction. *Journal of Wind Engineering and Industrial Aerodynamics*. 2002, vol. 90, no. 12-15, pp. 1403–1414.
7. BOWLER, Neill EH; PIERCE, Clive E; SEED, Alan. Development of a precipitation nowcasting algorithm based upon optical flow techniques. *Journal of Hydrology*. 2004, vol. 288, no. 1-2, pp. 74–91.
8. CHEUNG, P; YEUNG, HY. Application of optical-flow technique to significant convection nowcast for terminal areas in Hong Kong. In: *The 3rd WMO International Symposium on Nowcasting and Very Short-Range Forecasting (WSN12)*. 2012, pp. 6–10.
9. XINGJIAN, SHI; CHEN, Zhourong; WANG, Hao; YEUNG, Dit-Yan; WONG, Wai-Kin; WOO, Wang-chun. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In: *Advances in neural information processing systems*. 2015, pp. 802–810.
10. SHI, Xingjian; GAO, Zhihan; LAUSEN, Leonard; WANG, Hao; YEUNG, Dit-Yan; WONG, Wai-kin; WOO, Wang-chun. Deep learning for precipitation nowcasting: A benchmark and a new model. In: *Advances in Neural Information Processing Systems*. 2017, pp. 5617–5627.

11. *International meteorological vocabulary*. 2nd ed. Secretariat of the World Meteorological Organization, 1992. No. 182. ISBN 978-92-63-02182-3.

12. KRÁČMAR, Jan. *Meteorologické radiolokátory* [online]. Czech Hydrometeorological Institute [visited on 2018-11-16]. Available from: `http://portal.chmi.cz/files/portal/docs/meteo/rad/info_radar/index.html`.

13. LI, Ping-Wah; WONG, Wai-Kin; CHEUNG, Ping; YEUNG, Hon-Yin. An overview of nowcasting development, applications, and services in the Hong Kong Observatory. *Journal of Meteorological Research*. 2014, vol. 28, no. 5, pp. 859–876.

14. MATHIEU, Michaël; COUPRIE, Camille; LECUN, Yann. Deep multi-scale video prediction beyond mean square error. *CoRR*. 2015, vol. abs/1511.05440. Available from arXiv: `1511.05440`.

15. GOODFELLOW, Ian; POUGET-ABADIE, Jean; MIRZA, Mehdi; XU, Bing; WARDE-FARLEY, David; OZAIR, Sherjil; COURVILLE, Aaron; BENGIO, Yoshua. Generative adversarial nets. In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

16. MATHIEU, Michael; COUPRIE, Camille; LECUN, Yann. *Frame predictions extrapolated from UCF101 (Adv+GDL)* [online]. New York University – Computer Science Department, 2016 [visited on 2016-12-20]. Available from: `https://cs.nyu.edu/~mathieu/iclr2016.html`.

17. LECUN, Yann; BOSER, Bernhard E; DENKER, John S; HENDERSON, Donnie; HOWARD, Richard E; HUBBARD, Wayne E; JACKEL, Lawrence D. Handwritten digit recognition with a back-propagation network. In: *Advances in neural information processing systems*. 1990, pp. 396–404.

18. KRIZHEVSKY, Alex; SUTSKEVER, Ilya; HINTON, Geoffrey E. Imagenet classification with deep convolutional neural networks. In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

19. LI, Fei-Fei; JOHNSON, Justin; YEUNG, Serena. *Convolutional Neural Networks for Visual Recognition* [online]. Stanford University, 2018 [visited on 2018-11-12]. Available from: `http://cs231n.stanford.edu/`.

20. GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

21. RONNEBERGER, Olaf; FISCHER, Philipp; BROX, Thomas. U-net: Convolutional networks for biomedical image segmentation. In: *International Conference on Medical image computing and computer-assisted intervention*. 2015, pp. 234–241.

22. BENGIO, Yoshua; SIMARD, Patrice; FRASCONI, Paolo. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*. 1994, vol. 5, no. 2, pp. 157–166.

23. HOCHREITER, Sepp; SCHMIDHUBER, Jürgen. Long short-term memory. *Neural computation*. 1997, vol. 9, no. 8, pp. 1735–1780.

24. CHOLLET, François et al. *Keras* [`https://keras.io`]. 2015.

25. LIANG, Xiaodan; LEE, Lisa; DAI, Wei; XING, Eric P. Dual motion GAN for future-flow embedded video prediction. In: *IEEE International Conference on Computer Vision (ICCV)*. 2017, vol. 1.

26. IOFFE, Sergey; SZEGEDY, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*. 2015.

27. *Rušení meteorologických radiolokátorů CZRAD* [online]. Czech Hydrometeorological Institute [visited on 2018-12-04]. Available from: `http://radar4ctu.bourky.cz/Ruseni.html`.

28. ABADI, Martin et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems* [`https://www.tensorflow.org/`]. 2015.

29. NVIDIA CORPORATION. *CUDA Toolkit*. Version 9.0. Available also from: `https://developer.nvidia.com/cuda-toolkit`.

30. NVIDIA CORPORATION. *cuDNN*. Version 7.0. Available also from: `https://developer.nvidia.com/cudnn`.

31. ANACONDA, INC. *Anaconda*. Version 4.5.4. Available also from: `https://www.anaconda.com/`.

32. COLLETTE, Andrew. *HDF5 for Python* [`http://h5py.alfven.org`]. 2008.

33. HINTON, Geoffrey; SRIVASTAVA, Nitish; SWERSKY, Kevin. *Neural Networks for Machine Learning: Overview of mini-batch gradient descent* [`http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`].

34. ZEILER, Matthew D. ADADELTA: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*. 2012.

35. KINGMA, Diederik P; BA, Jimmy. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*. 2014.

36. SRIVASTAVA, Nitish; HINTON, Geoffrey; KRIZHEVSKY, Alex; SUTSKEVER, Ilya; SALAKHUTDINOV, Ruslan. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*. 2014, vol. 15, no. 1, pp. 1929–1958.

37. QUAN, Tran Minh; HILDEBRAND, David GC; JEONG, Won-Ki. Fusionnet: A deep fully residual convolutional neural network for image segmentation in connectomics. *arXiv preprint arXiv:1612.05360*. 2016.

Appendices

A More Examples of Full-Frame Merged Images

In this section, we show more examples of merged full-frame results as were discussed in Section 7.5. Each column shows the same situation. From top to bottom, our prediction, ground truth and the prediction by the Czech Hydrometeorological Institute are shown.

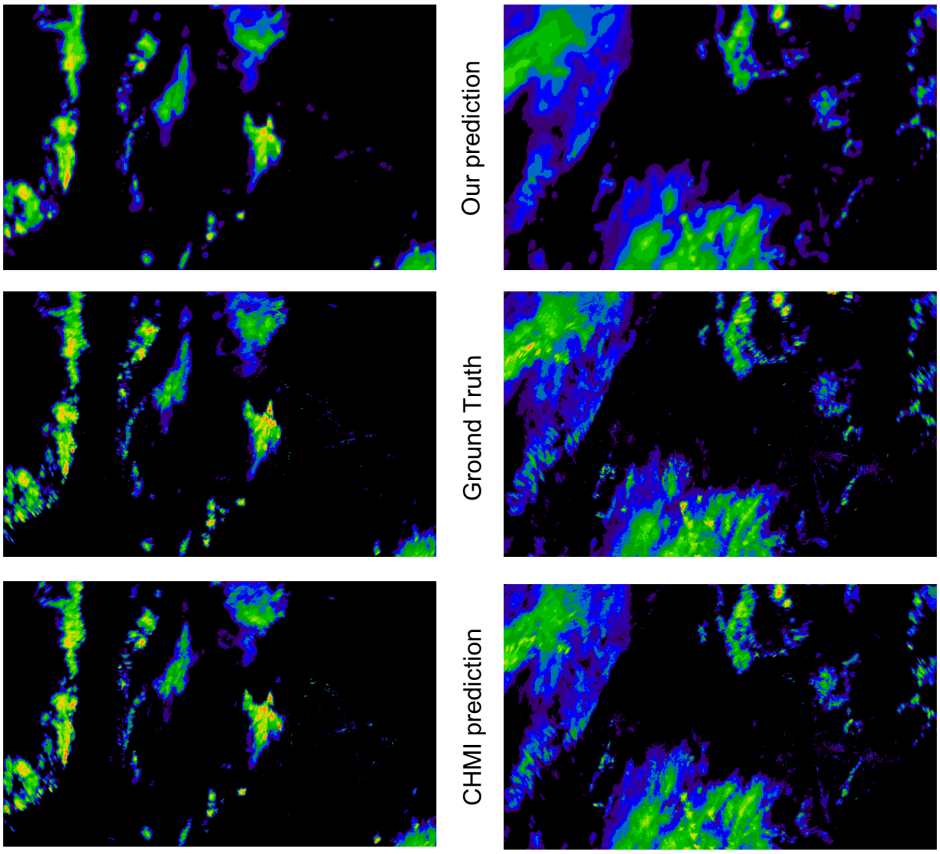


Figure A.1: Prediction by network Unitos4 compared with ground truth and CHMI prediction. *Source:* GT and CHMI prediction are from the Czech Hydrometeorological Institute [1].

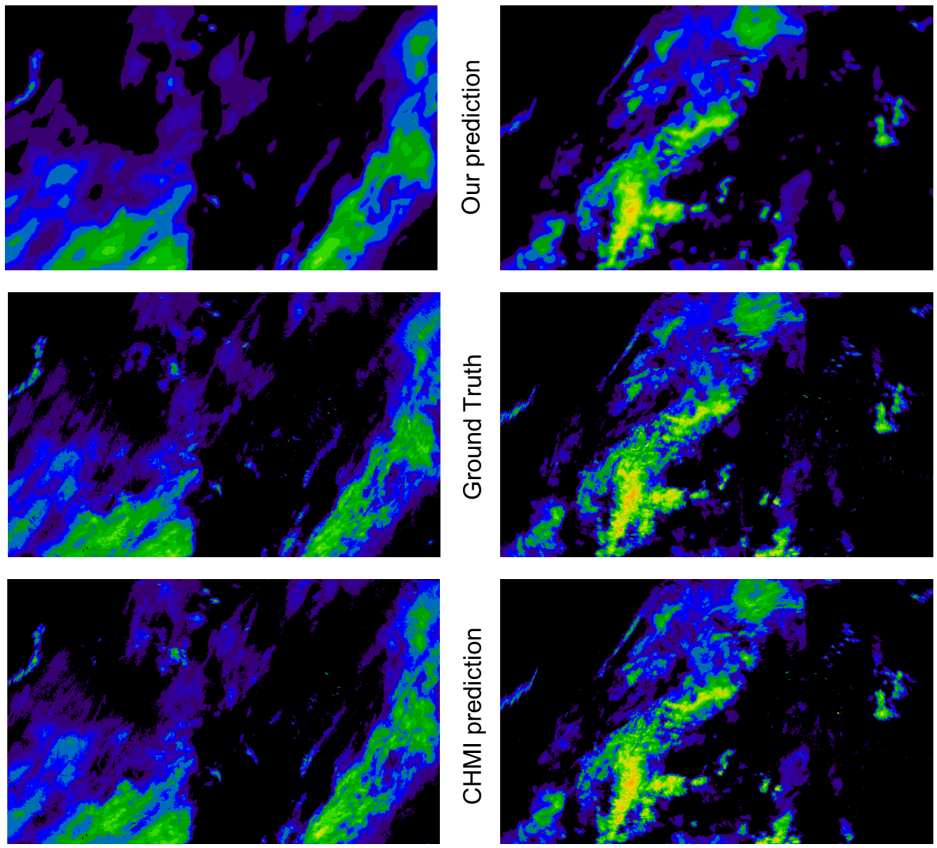


Figure A.2: Prediction by Unitos4 network compared with ground truth and CHMI prediction. *Source:* GT and CHMI prediction are from the Czech Hydrometeorological Institute [1].

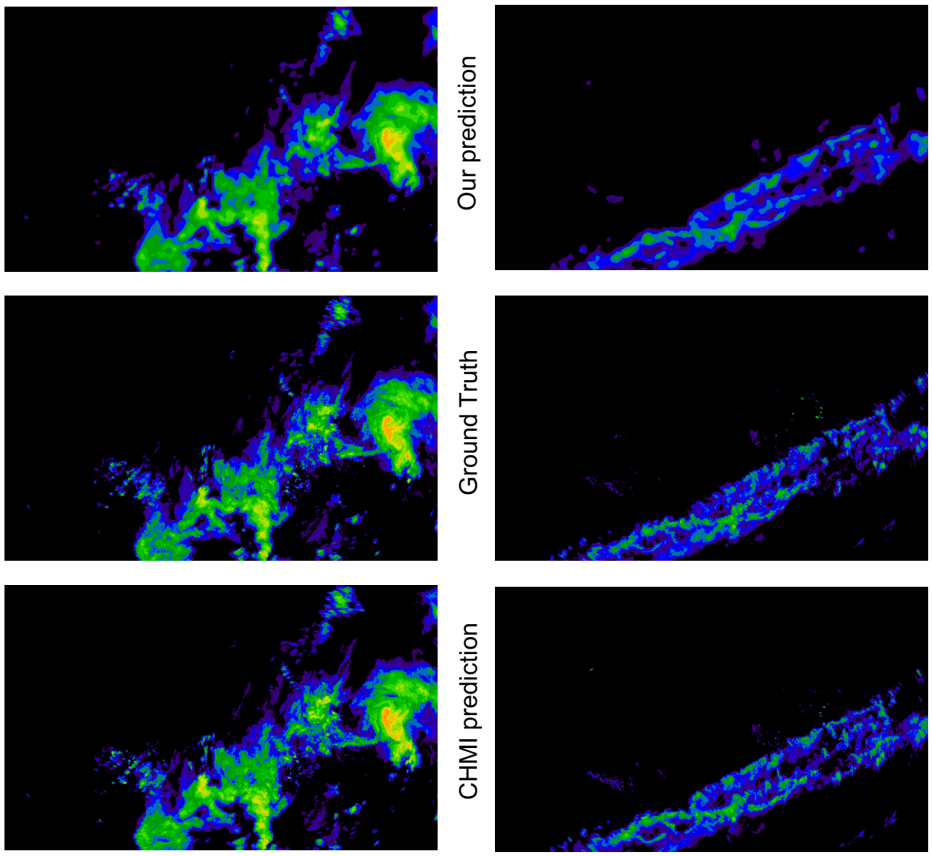


Figure A.3: Prediction by Unitos4 network compared with ground truth and CHMI prediction. *Source:* GT and CHMI prediction are from the Czech Hydrometeorological Institute [1].

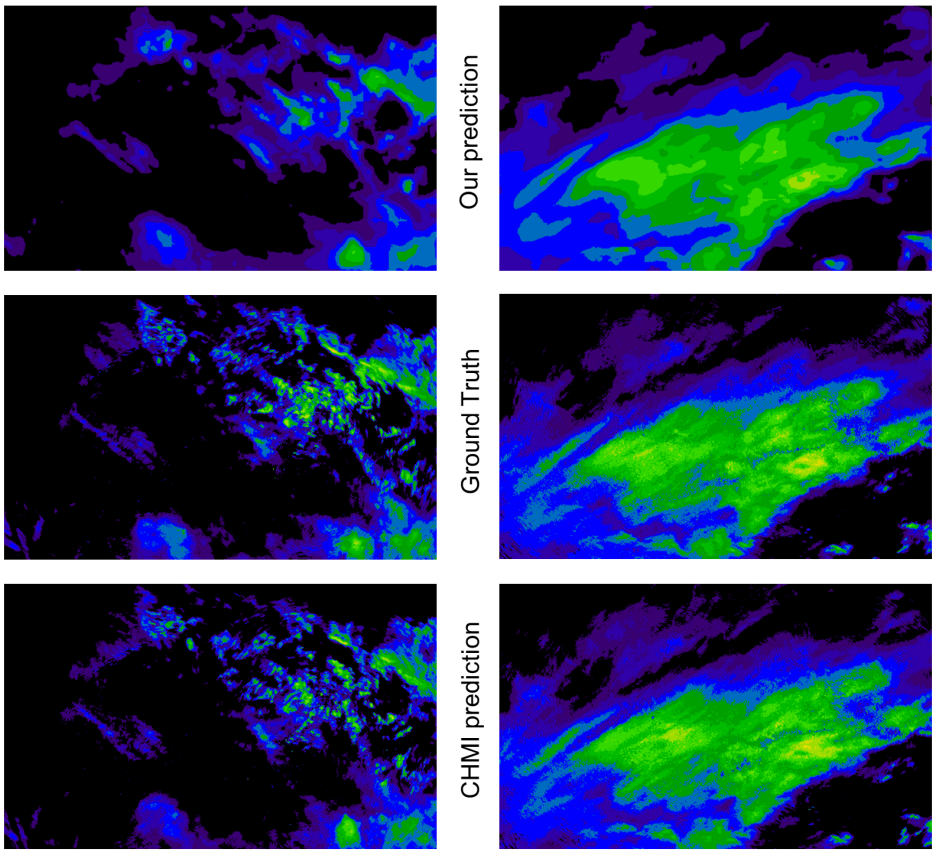


Figure A.4: Prediction by Unitos4 network compared with ground truth and CHMI prediction. *Source:* GT and CHMI prediction are from the Czech Hydrometeorological Institute [1].