FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

ASSIGNMENT OF BACHELOR'S THESIS

Title:	Interpolation and extrapolation of subsequent weather radar images
Student:	Matej Choma
Supervisor:	Ing. Jakub Bartel
Study Programme:	Informatics
Study Branch:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	Until the end of winter semester 2020/21

Instructions

The thesis aims to explore the possibilities of interpolating and extrapolating a sequence of weather radar images as an enhancement of nowcasting methods using machine learning algorithms.

1. Prepare dataset from images captured by weather radars.
2. Explore current machine learning methods for image recognition in terms of their use for video frame interpolation and extrapolation.
3. Based on these methods create a model for interpolation and extrapolation of sequences of weather radar images.
4. Use, analyse and comment results of created models. Compare the results to approaches that do not use machine learning.
5. Discuss further approaches in the field of weather nowcasting.

References

Will be provided by the supervisor.

Ing. Karel Klouda, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague February 25, 2019

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Interpolation and Extrapolation of Subsequent Weather Radar Images

Matej Choma

Department of Applied Mathematics

Supervisor: Ing. Jakub Bartel

May 16, 2019

Acknowledgements

I thank my supervisor, Ing. Jakub Bartel for his leadership, hours of consultations and insightful comments to both writing and execution of this thesis.

My sincere thanks also belong to the company Meteopress for the provided weather radar data and great motivation.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on May 16, 2019 ……

Czech Technical University in Prague

Faculty of Information Technology

Citation of this thesis

Choma, Matej. *Interpolation and Extrapolation of Subsequent Weather Radar Images*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2019.

Abstract

Recent advancement in the communication technologies creates an opportunity for the effective use of nowcasting – predicting the weather over short periods and communicating the predictions to the public in real time. Current nowcasting systems using weather radar images in the Czech Republic are built on traditional algorithms and numerical weather models. The objective of this work is to research the possibilities of using neural networks for processing of weather radar image sequences as an enhancement to the existing nowcasting methods.

I propose and test a convolutional neural network as a solution for two tasks – interpolation and extrapolation of a sequence of weather radar images. In both tasks, the proposed network achieves promising results. Quantitative comparison of my network and the currently used method COTREC ended in favour of the proposed network. The qualitative evaluation shows multiple benefits of my solution – the ability to capture the change of the radar echo intensity and shape. The results of this work create space for deeper research in this field.

Keywords weather radar images, Meteopress, image sequence interpolation, image sequence extrapolation, weather nowcasting, convolutional neural networks

Abstrakt

V dôsledku nedávneho pokroku v komunikačných technológiách vzniká príležitosť na efektívne využitie nowcastingu – poskytovanie krátkodobých predpovedí v reálnom čase. Súčasné nowcastingové systémy využívajúce radarové snímky v Českej republike sú založené na tradičných algoritmoch a numerických modeloch na predpovedanie počasia. Cieľom tejto práce je preskúmať možnosti využitia neurónových sietí na spracovanie sekvencií radarových snímok, z pohľadu vylepšenia existujúcich nowcastingových metód.

V práci navrhujem a testujem konvolučnú neurónovú sieť ako riešenie dvoch úloh – interpolácie a extrapolácie sekvencie radarových snímok. Navrhnutá sieť dosahuje pre obe úlohy sľubné výsledky. Kvantitatívne porovnanie mojej siete a metódy COTREC, ktorá sa aktuálne používa, skončilo v prospech siete. Kvalitatívne vyhodnotenie ukazuje viaceré výhody môjho riešenia – schopnosť zachytiť zmenu intenzity a tvaru radarového echa. Výsledky práce vytvárajú priestor pre ďalší výskum v tejto oblasti.

Klíčová slova radarové snímky zrážok, Meteopress, interpolácia sekvencie snímok, extrapolácia sekvencie snímok, weather nowcasting, konvolučné neurónové siete

Contents

Introduction **1**

Thesis's Objective **3**

1 Meteorological Background **5**
1.1 Weather Radars 6
1.1.1 Radar Networks in the Czech Republic 7
1.2 OPERA Programme 9
1.3 Nowcasting in the Czech Republic 9
1.3.1 COTREC 9

2 Machine Learning Background **11**
2.1 Convolutional Neural Networks 11
2.1.1 Convolutional Layer 11
2.1.2 Pooling Layer 13
2.1.3 Upsampling Layer 13
2.1.4 Encoder-decoder Architecture 13
2.2 Loss Functions 14
2.3 Improving Training 14
2.3.1 PReLu Activation Function 14
2.3.2 Residual Conections 14
2.3.3 Batch Normalization 15
2.4 SSIM 15

3 Image Sequence Processing and Related Work **17**
3.1 Tasks Definitions 17
3.1.1 Image Sequence Interpolation 17
3.1.2 Image Sequence Extrapolation 17
3.2 Optical Flow 18

3.3 Related Work to Image Sequence Interpolation 19
3.3.1 Learning Image Matching by Simply Watching Video . 19
3.3.2 Video Frame Interpolation via Adaptive Separable Convolution 19
3.3.3 Deep Video Frame Interpolation using Cyclic Frame Generation 20
3.4 Related Work to Image Sequence Extrapolation 21
3.4.1 Learning to Generate Long-term Future via Hierarchical Prediction 22
3.4.2 SDC-Net: Video prediction using spatially-displaced convolution 22

4 Dataset **25**
4.1 Source Images 25
4.2 Data Augmentation 25
4.3 Dataset for Interpolation Task 26
4.4 Dataset for Extrapolation Task 27
4.5 Implementation Details 27

5 Image Sequence Interpolation **29**
5.1 Architecture 29
5.2 Loss Function 30
5.2.1 VGG19 Loss 30
5.2.2 Combined Loss 33
5.3 Training 34
5.4 Results 35
5.5 Implementation Details 37

6 Image Sequence Extrapolation **39**
6.1 Multiple-step Extrapolation 39
6.2 Training 39
6.3 Results 41

Conclusion **43**
Outline of Future Work 44

Bibliography **45**

A Acronyms **49**

B Contents of enclosed DVD **51**

C Examples of Extrapolation of Weather Radar Image Sequences **53**

List of Figures

1.1 Coverage of the CZRAD radars. 7
1.2 Effect of the Earth curvature on the radar measurements. [10] . . 8
1.3 CAPPI weather radar image. 9

2.1 Diagram of a convolutional layer. 12
2.2 Receptive field. 12
2.3 Diagrams of pooling and upsampling layers. 13
2.4 SSIM comparison of images with various distortions. 16

3.1 Sequence of weather radar images. 18
3.2 Diagram of cycle consistency loss. [31] 21
3.3 Diagram of image generator. [34] 23

4.1 [580 × 294] weather radar image above the Czech Republic. 26
4.2 Colour coding of radar echo intensity. 26
4.3 Example from the interpolation dataset. 27
4.4 Example from the extrapolation dataset. 28

5.1 The architecture of the CNN. 31
5.2 The architecture of the convolutional and upsample block. 31
5.3 Layers of the VGG19 architecture. 32
5.4 Examples of learning with $\mathcal{L}_F$ 33
5.5 Examples of learning with $\mathcal{L}_C$ 34
5.6 Plot of loss during the training. 35
5.7 Output of the network after 50^{th}, 70^{th} and 85^{th} epoch. 36
5.8 Examples of image sequence interpolation. 37

6.1 Plot of loss during the training. 40
6.2 Output of the network after 40^{th} and 80^{th} epoch. 40
6.3 Example of image sequence extrapolation. 42

C.1 Extrapolation example 1 54
C.2 Extrapolation example 2 55
C.3 Extrapolation example 3 56

List of Tables

4.1 Shapes of the datasets for the interpolation task. 27
4.2 Shapes of the datasets for the extrapolation task. 28

5.1 Output shapes of convolutional blocks. 30
5.2 Values of average SSIM index on the validation dataset for various $\mathcal{L}_F$ loss functions. 32
5.3 Values of average SSIM index on the validation dataset for various $\mathcal{L}_C$ loss functions. 34
5.4 Values of average SSIM index on the test dataset for the interpolation task. 36

6.1 Values of average SSIM index on the test dataset for the extrapolation task. 41

Introduction

Climate and weather as a symptom of it shape human civilization. Storms, severe wind, floods, droughts and many others are caused by weather and put human lives and property to risk. Weather determines how farmers will take care of crops, how much electrical energy will solar, hydroelectric and wind powerplants produce and affects a lot of human decisions in everyday life. As it is not possible to change the weather as needed, it is important to know, what will the weather be like and adapt to it.

Images of precipitation from weather radars are a popular type of weather information. They display the exact area covered by precipitation and capture its motion in the past. The knowledge of the future precipitation motion in a sufficient time can help to protect humans and property from storms and heavy rainfalls, while there can still be done a lot to enhance the user experience when informing about the past development. This work uses machine learning techniques to work with radar images, captured above the area of the Czech Republic, as frames of a video.

Machine learning algorithms for two different sequence processing methods will be explored in this thesis. The first method is the interpolation of the precipitation images sequence – estimation of images between actual images of the sequence. While the sequence obtained from weather radars is sparse, better-looking information can be presented using this technique. The second type of sequence processing is extrapolation – estimation of future images of the sequence, which will be used as a prediction of the precipitation development.

The following two chapters introduce the reader to the meteorological and machine learning background. The interpolation and extrapolation tasks are defined in Chapter 3 alongside the research of current state-of-the-art methods.

Chapter 4 describes the creation of the dataset from the weather radar images. Machine learning models for the tasks are introduced, trained and evaluated in Chapters 5 and 6. Chapter 6 also contains a comparison of my model and the method COTREC for the extrapolation task.

Thesis's Objective

The objective of this thesis is to explore the possibilities of interpolating and extrapolating a sequence of weather radar images as an enhancement of nowcasting methods using machine learning algorithms.

The goal of the theoretical part is to introduce the reader to the current weather nowcasting methods and necessary machine learning concepts. This part also covers research of the current state-of-the-art machine learning methods for image recognition in terms of their use for video frame interpolation and extrapolation.

The practical part will begin with the creation of dataset from weather radar images from the area above the Czech Republic. The objective of this part is to design and implement models for interpolation and extrapolation of sequences of weather radar images. The models will be trained and tested on the created dataset and results are to be analysed and compared to methods that do not use machine learning. The goal of the last part of the thesis is to outline possible future work, how to further improve the results of interpolation and extrapolation of weather radar image sequences.

Chapter 1

Meteorological Background

The weather has always affected human lives. For a long time, favourable weather was a question of survival for a society. The whole production of food depended on it as crops, animals and nature as a whole need specific weather conditions to prosper. Even small fluctuations in weather like heavy rainfalls or low temperatures exposed human lives to great risks.

Much has not changed nowadays. Enough rainfall, sunshine, good temperatures and no severe weather are still necessary to feed humanity. The climate change that is one of the reasons for migration in recent years [1] is a good example. Even though people can protect themselves from various types of weather, it still causes many natural disasters (floods, tornadoes,…) that endanger human lives and damage property. In addition to these, nowadays weather affects all types of traffic (especially air-traffic), production of electricity from multiple energy sources (wind, hydroelectric and solar power plants), planning and organisation of outdoor events, etc.

Based on local observations, humans were creating weather lore to make short-term predictions. With the arrival of meteorology (the science of the atmosphere [2]) physical quantities in the atmosphere started to be measured, and it became possible to make longer-term and more rigorous weather forecasts.

One specific type of weather forecasting is nowcasting. It is defined in [3] as *"forecasting with local detail, by any method, over a period from the present to 6 hours ahead, including a detailed description of the present weather"*. As described in [4], multiple challenges need to be overcome to make nowcasting effective. Firstly, there is a need for a large amount of high-density weather information. Secondly, this information needs to be communicated to the target group (public, military, air-traffic, …) in real time. And finally, the target group has to adapt to the acquired information. Arise of the compu-

tational power, use of mobile devices and social media in recent years answer these challenges and make the development of nowcasting systems up-to-date topic.

1.1 Weather Radars

Radar (from acronym *RAdio (Aim) Detecting And Ranging*) is a system for detection of objects. The principle of radar operation is similar to sound-wave reflection – an effect of hearing an echo when shouted against a sound-reflecting object, for example in a cave. The general direction of reflecting object can be determined from the echo, and even distance can be estimated, given the knowledge of the speed of the sound. [5, 6]

The radar emits electromagnetic waves, which travel with the speed of light, in a specific direction. When these waves meet an electrically leading surface, a portion of the energy is reflected/scattered in all directions. Radar receives the reflected energy and can calculate the distance of the reflecting object based on the time elapsed since the emission of the pulse. Electromagnetic waves travel in a straight line through space, so the distance and the direction (azimuth and elevation) of transmitted pulse exactly determine the position of the reflecting object. A signal containing this information is also called radar echo. [5, 6]

Water and therefore, precipitation is conductive and reflects electromagnetic waves. When measuring precipitation, radar is slowly moving in a constant elevation. Emitted electromagnetic beams have some width and radar can send dozens of pulses and measure reflected energy, during each degree of rotation. To obtain a full 3D image of precipitation, these scans are run in multiple different elevations. [7]

Important parameters of weather radars are together related wavelength ($[\lambda] = m$) and frequency ($[f] = Hz = \frac{1}{s}$)

$$\lambda = \frac{c}{f}, \tag{1.1}$$

where c ($[c] = \frac{m}{s}$) stands for the speed of light, which is constant. Large wavelength pulses have low frequency and cannot detect weak precipitation, while ones with smaller wavelength and higher frequency reflect almost all of their energy in heavy rain or storm and cannot well detect objects behind it. Modern radars can use dual polarisation for detecting the shape of the object and deciding about the type of precipitation. Doppler effect [8] is used to describe the movement of the detected object. [7]

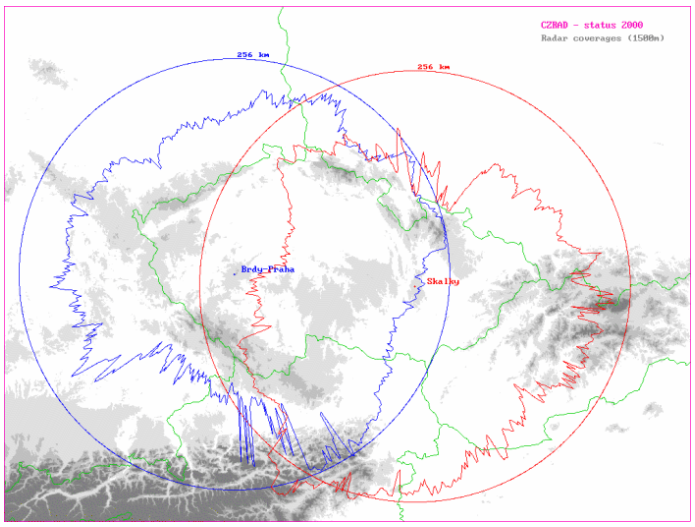


Figure 1.1: Coverage of the CZRAD radars. Circles denote the theoretical ranges. The other lines limit the real coverage of the radar, based on various physical aspects. [9]

1.1.1 Radar Networks in the Czech Republic

The precipitation over the Czech Republic is monitored with two radar networks located inside the borders of the country. According to [7], this area is partially covered also with German and Slovak radars.

1.1.1.1 Czech Weather Radar Network

The Czech Weather Radar Network (CZRAD) is the radar network of the Czech Hydrometeorological Institute (CHMI). It consists of two C-band radars with the frequency of 5 GHz. Radars with this frequency achieve good precipitation measurements, but the signal can interfere with WiFi networks that are also run on 5 GHz. Each of the CZRAD radars can cover up to 256 km range (Figure 1.1). In this case, the range is not limited by the power of the radar but rather by the curvature of the Earth (Figure 1.2), as the most valuable data is about the lowest parts of the atmosphere. [9, 7]

Precipitation data is obtained at locations Skalky u Protivanova and Prague-Brdy and combined to a publicly available full volume scan every 10 minutes. Scans are run in multiple elevations from top to bottom. The data about precipitation just above the Earth surface is therefore as actual as possible, while information about the top levels of the atmosphere is already a few minutes old when delivered to users. Processed full volume images are then published on the CHMI Nowcasting webportal[1]. [7]

[1] `http://portal.chmi.cz/files/portal/docs/meteo/rad/inca-cz/short.html`

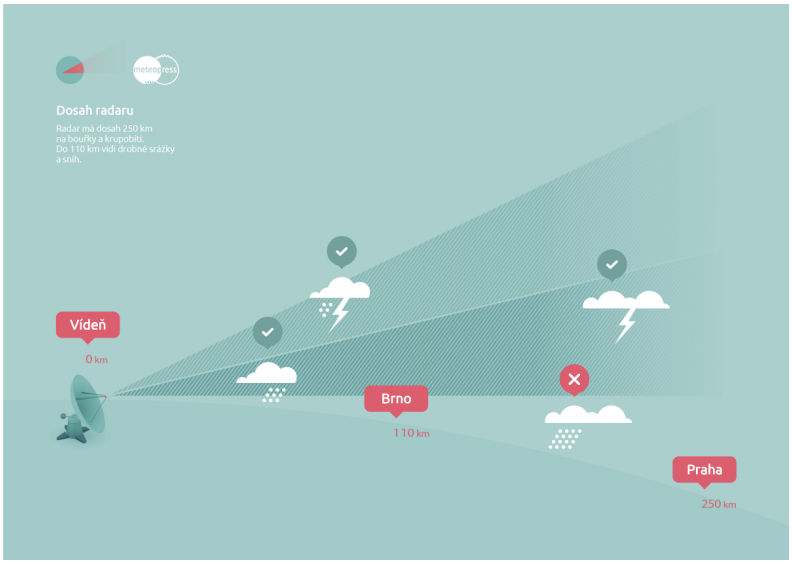


Figure 1.2: Effect of the Earth curvature on the radar measurements. [10]

CZRAD radars have a horizontal resolution of 1×1 km, a vertical resolution of 0.5 km and capture 256 degrees of precipitation intensity. Multiple products are created from the data. The two most common are Radar Reflexivity MAX Z and CAPPI. The Radar Reflexivity MAX Z images are created taking the maximum precipitation intensity through the whole height. The CAPPI displays precipitation in a constant height above the sea level. The number of precipitation intensity degrees in both types of images is reduced to 16. Published CAPPI image can be seen in Figure 1.3. [9]

1.1.1.2 Meteopress Radar Network

There is a new radar network emerging in the last decade over Central Europe. In the last two years, the company Meteopress has placed and is operating radars in the Czech Republic and Slovakia. They use X-band radars of their production that are cheaper to build but with lower power output and different parameters than the CZRAD C-band radars. Meteopress radars are operating on the frequency around 10 GHz and therefore cannot detect well objects behind a storm or heavy rain. They approach this challenge with more dense radar network consisting of 6 radars with plans to add two more. [10]

The main benefit of Meteopress's radars, when compared to CZRAD, is the resolution. Their radars can create a scan in 1-minute interval with a horizontal resolution from 150×150 m to 250×250 m. Precipitation images are displayed at `https://www.meteopress.cz/radar/` during heavy rainfalls or storms. [10]

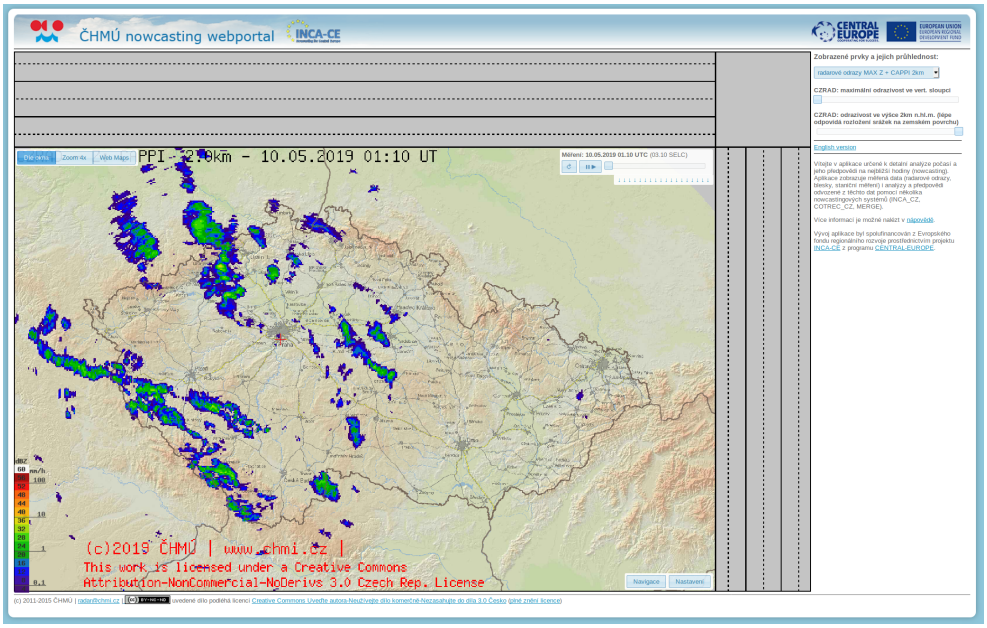


Figure 1.3: Screenshot of CZRAD radar image of type CAPPI, published on the CHMI Nowcasting webportal. [11]

1.2 OPERA Programme

"OPERA is the radar programme of EUMETNET". [12] EUMETNET is the Conference of the National Meteorological Services, formed primarily to help cooperation and collaboration among its members. EUMETNET also represents the National Meteorological services externally.

The key achievement of the OPERA programme is a high-quality composite of weather radar images across the majority of Europe region, produced nearly in real time. Data from the CZRAD radar network and German and Slovak weather radars are also included. [12]

1.3 Nowcasting in the Czech Republic

To the best of my knowledge, the only nowcasting system in the Czech Republic using weather radar images is the one created by the CHMI. According to [11], a nowcasting model based on the COTREC method is used.

1.3.1 COTREC

COTREC [13] is a nowcasting method for extrapolation of the radar echo (signal reflected back to the radar by an object). It is built on the TREC method and improves it in two ways. [14] shows that in 2007, the CHMI im-

plementation of COTREC method outperformed the numerical Aladin model during the first 3 hours of prediction. One case where COTREC lags is that it cannot capture well growth or decay of radar echo.

1.3.1.1 TREC Method

TREC, as explained in [15], assumes that precipitation motion stays constant over a short period, and there is no growth or decay of the radar echo intensity. TREC takes two consecutive weather radar images on the input and calculates TREC vectors – *"translation vectors of radar echo patterns"* [15], also called shift or wind motion vectors – from them.

The whole radar domain (area covered by the radar) is split to rectangle boxes of the same size. For each of the boxes, a TREC vector is found shifting the box from the second image to every possible box position in the first and finding the one that is the most similar. Use of the Pearson correlation coefficient as the similarity metric is proposed in the [15], but also metrics like ℓ_1 or ℓ_2 norms can be used.

Feature frames are synthesised pixel-by-pixel from the most recent observed input image. Each pixel in the output image is shifted by the corresponding motion vector backwards to obtain an originating point in the input image, and the radar echo intensity of this point is used for the target one. It is assumed that the TREC vector for a box corresponds to the motion vector of the point in the middle of the box. For other points, interpolation of the TREC vectors from four nearest boxes is used.

1.3.1.2 COTREC Improvements to TREC Method

Calculation of the TREC vectors often leads to inconsistent or noisy results [15]. This behaviour is caused primary by ground clutter (unwanted radar echoes reflected from the surface) or shielding of the radar beams. Proposed solution [15] replaces incorrect vectors with the average of the neighbouring ones. As the incorrect vectors are considered vectors with zero velocity or the ones that deviate more than 25° from the direction of the mean of the neighbouring vectors.

In the second step, the whole motion field is smoothed, with two requirements in mind. TREC-derived motion vectors should be continuous, and the outcome must be as similar as possible to the original motion field. The exact process is beyond the scope of this thesis and can be found in [15].

For comparisons in Section 6 will be used the implementation of COTREC by the company Meteopress.

Chapter 2

Machine Learning Background

This chapter covers prerequisite concepts needed in this thesis. It is assumed that reader is familiar with the concept of feedforward neural networks. The topic can be studied in the first chapter of [16].

2.1 Convolutional Neural Networks

Convolutional neural networks (CNN) [17] are neural networks that expect the input to be a 3D tensor (with dimensions width, height and depth), typically an image. Neurons of a CNN are arranged as well. CNNs proposed in this thesis will be fully convolutional networks (FCN) and primary use the following layers: convolutional layer, pooling layer and upsampling layer.

FCNs [18] are a special type of CNNs where every layer is a filter applied over each region of the input according to stride. One of the benefits of such a network is that it can naturally process the input of any size.

2.1.1 Convolutional Layer

The core of CNNs are convolutional layers [17]. The learnable parameters are 3D convolutional filters that are during the forward pass slid across the width and height of the input tensors. At each position, a dot product of filter and input is computed that together form a 2D activation map. Typically a set of filters is used in a convolutional layer and activations are concatenated along the depth dimension to output a 3D tensor. Intuitively, the filters activate when they see some visual feature represented by the filter weights. For example, it can be an edge in the first layer or some more complex shape at later ones. A simple diagram of convolutional layer is in Figure 2.1.

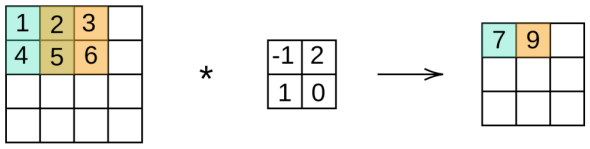


Figure 2.1: Diagram of a convolutional layer with input dimensions $[4 \times 4 \times 1]$, filter dimensions $[2 \times 2 \times 1]$ and stride of size $(1, 1)$.

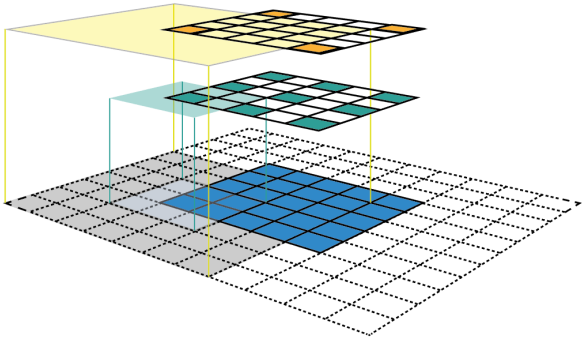


Figure 2.2: Visualization of a receptive field of two consecutive convolutional layers with $(3, 3)$ filter size, $(2, 2)$ stride and zero padding of size 1. [19]

2.1.1.1 Size of the Output and Number of Trainable Parameters

Consider input to a convolutional layer with size $[W_i \times H_i \times D_i]$ and the layer consisting of c filters, each with size $[w \times h \times D_i]$ (depth need to be same as the one of the input). The filter is moved with a stride of (W_s, H_s).

The height of the output H_o [17] is then computed as

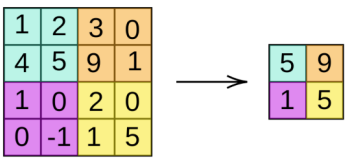
$$H_o = \frac{H_i - h + 2P}{H_s} + 1. \tag{2.1}$$

The width of the output W_o is computed analogically. The input can be padded with P zeros around the borders to make the numerator divisble by stride. The output depth is c – the number of filters in the layer. To summarize, the size of the output is $[W_o \times H_o \times c]$.

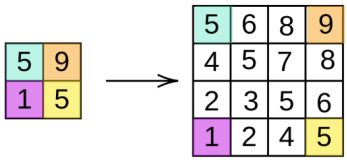
The number of parameters learned by the network is $w \cdot h \cdot D_i \cdot c$.

2.1.1.2 Receptive Field

Receptive field [19] of a convolutional layer is a region in the input space that affected a particular activation. The receptive field of the first convolutional layer has the same size as the filter. In later layers, the receptive field is a union of receptive fields of seen activations by the filter (Figure 2.2).



(a) Diagram of a max pooling layer with $(2, 2)$ filter and $(2, 2)$ stride.



(b) Diagram of upsampling layer with scale 2 and bilinear interpolation.

Figure 2.3: Diagrams of pooling and upsampling layers.

2.1.2 Pooling Layer

Pooling layer [17] is often added in-between convolutional layers to reduce the spatial size. It forces the network to keep just the important information and hence control the overfitting. Filter of size $w \times h$ is reduced to one number and the filter is slid across the width and height of the input with stride (W_s, H_s). Depth stays unchanged. The most common setting is the pooling layer with $(2, 2)$ filter and $(2, 2)$ stride (Figure 2.3a).

The actual pooling operation can be realised with multiple functions. The most used are max pooling (replacing numbers in the filter with a maximum of them) and average pooling (replacing numbers in the filter with mean of them). Pooling layer has no learnable parameters.

2.1.3 Upsampling Layer

Upsampling layer [20] is used to reconstruct the image from the spatially compressed output of pooling layers. The input to the layer is scaled-up interpolating the in-between points from the input ones (Figure 2.3b). Upsampling layer has no learnable parameters.

2.1.4 Encoder-decoder Architecture

CNNs outputing 2D tensors [21, 22, 23, 24] often share similar architecture (an example can be found in Figure 5.1). It consists of two parts:

1. **encoder** part – combination of convolutional and pooling layers is used to extract features from the input and downscale the spatial size

2. **decoder** part – combination of convolutional and upsample layers is used to reconstruct the spatial information

2.2 Loss Functions

Loss function [16] quantifies how well a machine learning model approximates data in the training dataset and guides the training of it.

2.2.0.1 L1 Loss

L1 loss $\mathcal{L}_1$, also called mean absolute error (MAE) as defined in [20], is a loss function built on a per-pixel colour difference basis. It is built on the ℓ_1 vector norm between interpolated image $\widehat{\mathbf{I}}$ and ground truth image $\mathbf{I}$:

$$\mathcal{L}_1(\mathbf{I}, \widehat{\mathbf{I}}) = \frac{1}{N}||\mathbf{I} - \widehat{\mathbf{I}}||_1, \tag{2.2}$$

where N denotes the number of elements in the input tensor.

2.2.0.2 MSE Loss

Mean squared error (MSE) loss $\mathcal{L}_{MSE}$, as defined in [20], is analogically to $\mathcal{L}_1$ a loss function built on the squared ℓ_2 vector norm:

$$\mathcal{L}_{MSE}(\mathbf{I}, \widehat{\mathbf{I}}) = \frac{1}{N}||\mathbf{I} - \widehat{\mathbf{I}}||_2^2, \tag{2.3}$$

where N denotes the number of elements in the input tensor.

2.2.0.3 Perceptual Loss

Perceptual loss [25] is a loss function based on a feature extractor, that is comparing two images based on the high-level features, rather than per-pixel. This loss should compare the input images on a basis similar to a human comparison.

2.3 Improving Training

2.3.1 PReLu Activation Function

Activation functions [16] are added to the network to model nonlinearity. An often used one, when working with images is rectified linear unit $\mathrm{ReLU}(x) = \max(0, x)$. The PReLU activation [20] is based on the ReLU and has following form, where a is a learnable parameter:

$$\mathrm{PReLU}(x) = \max(0, x) + a \cdot \min(0, x). \tag{2.4}$$

2.3.2 Residual Conections

In [26], authors propose the use of residual connections to help train deep networks. Using these connections, the output of some layer is summed with

the output of some previous layer with the same shape before feeding forward the activations to the next layer. These connections were used in [21] to maintain fine-grained image details during the decoding in an encoder-decoder CNN.

2.3.3 Batch Normalization

During training, each mini batch is sampled from the same training set, but they have different distributions which force the network to constantly adapt to a new distribution. Batch normalisation layer [27] normalises, scales and shifts the activations of the mini batch to reduce this problem. It is reported in the paper that including batch normalisation layers both speed up the training and reduce overfitting.

Batch normalization is applied to each dimension $x^{(k)}$ of the input x separately. Firstly, the input is normalized:

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}, \tag{2.5}$$

where $\mathrm{E}[\cdot]$ and $\mathrm{Var}[\cdot]$ respectively denotes expected value and variance over the trainin dataset. $\widehat{x}^{(k)}$ is then shifted with learned parameters $\gamma^{(k)}$ and $\beta^{(k)}$:

$$y^{(k)} = \gamma^{(k)}\widehat{x}^{(k)} + \beta^{(k)}. \tag{2.6}$$

2.4 SSIM

Structural Similarity (SSIM) Index as defined in [28] is a metric that measures the quality of an image compared to the ground truth image. It is based on the philosophy that humans pay the most attention to the structural information of an image when they judge about its quality. Therefore, the classical metrics like ℓ_1 norm or MSE are a little bit short-handed in image quality assessment. An image with little shifted colours can have the same MSE as the same image with noise instead (Figure 2.4), even though the one with noise is for humans much more distorted.

SSIM measures the similarity of two images $\mathbf{x}$ and $\mathbf{y}$ in three separate comparisons: luminance, contrast and structure.
Luminance is estimated as the mean intensity of the pixels x_i

$$\mu_x = \frac{1}{N}\sum_{i=1}^{N} x_i. \tag{2.7}$$

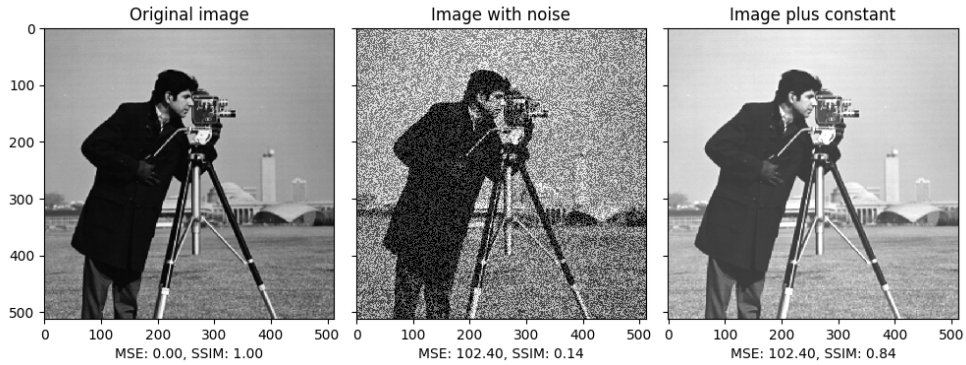


Figure 2.4: Comparison of images with various distortions, but same MSE. The one with noise is for humans more distorted than the shifted one, which is also shown by the SSIM value. [29]

Luminances μ_x and μ_y are compared using the function $l(\mathbf{x}, \mathbf{y})$, where C_1 is a constant:

$$l(\mathbf{x}, \mathbf{y}) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1}. \tag{2.8}$$

Contrast is estimated as the standard deviation

$$\sigma_x = \Big(\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu_x)^2\Big)^{\frac{1}{2}} \tag{2.9}$$

and σ_x and σ_y are comapred with function $c(\mathbf{x}, \mathbf{y})$, where C_2 is a constant:

$$c(\mathbf{x}, \mathbf{y}) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2}. \tag{2.10}$$

Structure comparison $s(\mathbf{x}, \mathbf{y})$ is defined as follows:

$$s(\mathbf{x}, \mathbf{y}) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3}. \tag{2.11}$$

Correlation coefficient σ_{xy} between images $\mathbf{x}$ and $\mathbf{y}$ can be estimated as:

$$\sigma_{xy} = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu_x)(y_i - \mu_y). \tag{2.12}$$

Metrics are combined into one SSIM index as following, where $C_3 = C_2/2$:

$$SSIM(\mathbf{x}, \mathbf{y}) = l(\mathbf{x}, \mathbf{y}) \cdot c(\mathbf{x}, \mathbf{y}) \cdot s(\mathbf{x}, \mathbf{y}) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}. \tag{2.13}$$

Implementation of SSIM from the `scikit-image`[2] library is later used in this work.

[2] `https://scikit-image.org`

Chapter 3

Image Sequence Processing and Related Work

The review of the related work using machine learning algorithms for image sequence processing is done in this chapter. It will be reffered to image sequence interpolation/extrapolation also as video frame interpolation/extrapolation in this thesis.

3.1 Tasks Definitions

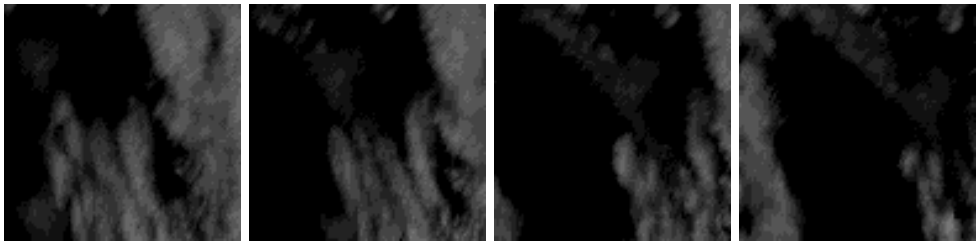
Both the image sequence interpolation and extrapolation are regression learning problems. An image $\mathbf{I} \in \mathbb{R}^{I \times J}$ is considered as a matrix of real numbers.

3.1.1 Image Sequence Interpolation

In the image sequence interpolation problem, there is a triplet $(\mathbf{I}_{t_1}, \mathbf{I}_{t_2}, \mathbf{I}_{t_3})$ of subsequent images capturing a scene in times (t_1, t_2, t_3). In case of this work, these are weather radar images of precipitation. It is assumed that time steps Δ_t between images are consistent, $\Delta_t = t_2 - t_1 = t_3 - t_2$. The task is to estimate the image $\widehat{\mathbf{I}_{t_2}}$ from input images $\mathbf{I}_{t_1}$ and $\mathbf{I}_{t_3}$ that will be as simillar as possible to the real (ground truth) image $\mathbf{I}_{t_2}$.

3.1.2 Image Sequence Extrapolation

The goal of image sequence extrapolation task is to estimate future images that fit to a sequence of observed ones. The input sequence consists of N weather radar images $(\mathbf{I}_{t_1}, \mathbf{I}_{t_2}, \ldots, \mathbf{I}_{t_N})$, captured in times $(t_1, t_2, \ldots, t_N)$. The time step Δ_t is a constant $t_i - t_{i-1} = \Delta_t,\ i \in \{2, 3, \ldots, N\}$. The task is to predict a sequence of M future images $(\mathbf{I}_{N+1}, \mathbf{I}_{N+2}, \ldots, \mathbf{I}_{N+M})$ with the same Δ_t, so $t_i - t_{i-1} = \Delta_t,\ i \in \{N+1, N+2, \ldots, M\}$.



(a) $\mathbf{I}_{t_1}$ (b) $\mathbf{I}_{t_2}$ (c) $\mathbf{I}_{t_3}$ (d) $\mathbf{I}_{t_4}$

Figure 3.1: Sequence of weather radar images.

3.2 Optical Flow

Optical flow, or flow in general, is a common term when talking about image sequence interpolation/extrapolation. The optical flow [30] estimates the 2D motion of image points between subsequent images. It is by definition the *"apparent motion of the brightness pattern."* [30] So, if lightning in the scene changes, there will be optical flow even though there is no motion. The optical flow from image $\mathbf{I}_0$ to $\mathbf{I}_1$ will be noted as $\mathbf{F}_0 \in \mathbb{R}^{2\times I\times J}$ and it is a set of vectors (u, v) associated with each pixel $\mathbf{x} = (x, y)$ of the image that captures the motion of that point between the two images.

3.2.0.1 Frame Interpolation with Optical Flow

One of the algorithms for calculating the interpolated image from the optical flow (frame synthesis) is described in [30]. This algorithm is used for evaluating optical flow estimation on the frame interpolation in the Middlebury benchmark [30]. The algorithm has four steps:

1. Flow $\mathbf{F}_{0.5}$ at time $t = 0.5$ is obtained by forward-warping $\mathbf{F}_0$ in such a way that:
$$\mathbf{F}_t(\mathbf{x} + t \cdot (\mathbf{F}_0(\mathbf{x}))) = \mathbf{F}_0(\mathbf{x}). \tag{3.1}$$
Vectors are assigned to every pixel which is in 0.5 radius from $\mathbf{x} + 0.5 \cdot (\mathbf{F}_0(\mathbf{x}))$. In case that multiple vectors are assigned to one pixel, the most fitting one, the one that minimise equation 3.2 is chosen.
$$|\mathbf{I}_0(\mathbf{x}) - \mathbf{I}_1(\mathbf{x} + \mathbf{F}_0(\mathbf{x})| \tag{3.2}$$

2. Holes in $\mathbf{F}_{0.5}$ are filled according to a closer unspecified outside-in filling strategy.

3. Occlusion masks O_0 and O_1, denoting which pixels from one image are not visible in the other, are estimated. $\mathbf{F}_1$ is calculated using equation

3.1 and $O_1(\mathbf{x}) = 1$ (pixel $\mathbf{x}$ from image $\mathbf{I}_1$ is not visible in $\mathbf{I}_0$) for every pixel $\mathbf{x}$ that has no flow vector assigned in $\mathbf{F}_1$. The other way $O_0(\mathbf{x}) = 1$ for every pixel $\mathbf{x}$ where:

$$|\mathbf{F}_0(\mathbf{x}) - \mathbf{F}_1(\mathbf{x} + \mathbf{F}_0(\mathbf{x}))| > 0.5. \tag{3.3}$$

4. For every pixel $\mathbf{x}$ in the interpolated frame $\mathbf{I}_{0.5}$ the corresponding pixels in $\mathbf{I}_0$ and $\mathbf{I}_1$ are calculated as $\mathbf{x}_0 = \mathbf{x} - 0.5 \cdot \mathbf{F}_{0.5}(\mathbf{x})$ and $\mathbf{x}_1 = \mathbf{x} + 0.5 \cdot \mathbf{F}_{0.5}(\mathbf{x})$. If according to occlusion masks both of the pixels are visible then

$$\mathbf{I}_{0.5}(\mathbf{x}) = 0.5 \cdot \mathbf{I}_0(\mathbf{x}_0) + 0.5 \cdot \mathbf{I}_1(\mathbf{x}_1). \tag{3.4}$$

Otherwise, the value of visible pixel is used.

3.3 Related Work to Image Sequence Interpolation

For a long time, image sequence interpolation was taken as a problem of estimating optical flow and synthesising interpolated frame from the optical flow and input images. However, a lot of work on video frame interpolation using neural networks was conducted in the last three years. First, to do so were Long et al. [21] in 2016. An interesting method that merges optical flow estimation and image synthesis into one step was introduced by Niklaus et al. [22] in 2017. Liu et al. [31] described the current method of state-of-the-art.

3.3.1 Learning Image Matching by Simply Watching Video

Long et al. were first to use a deep convolutional neural network directly for video frame interpolation in [21]. They twisted the traditional view on the interpolation – not estimate the optical flow to calculate interpolated frame but rather directly estimate the interpolated frame to calculate optical flow. The interpolated frame is estimated by a fully convolutional neural network that is trained on triplets of consecutive frames from widely available video data. Afterwards, based on the back-propagation, they compute for each pixel a gradient of its value with respect to every input pixel. The pixels from input images that affect the pixel the most are taken to calculate the motion field.

3.3.2 Video Frame Interpolation via Adaptive Separable Convolution

In [22] Niklaus et al. merge estimation of optical flow and intermediate frame synthesis into a single end-to-end trainable convolutional network (all parameters are trained simultaneously with one loss function). A pixel value at position (x, y) of interpolated frame $\widehat{\mathbf{I}_{0.5}}$ is calculated as follows:

$$\widehat{\mathbf{I}_{0.5}}(x, y) = K_0(x, y) * \mathbf{P}_0(x, y) + K_1(x, y) * \mathbf{P}_1(x, y). \tag{3.5}$$

$K_t(x, y) \in \mathbb{R}^{N \times N}$ denotes a 2D convolutional kernel for pixel (x, y) in the input frame $\mathbf{I}_t$ which is convolved with a patch $\mathbf{P}_t(x, y)$ from this image centered at (x, y). To dramatically reduce amount of used memory, authors propose to use two 1D convolutional kernels $(K_{t,v}, K_{t,h})$ and estimate the 2D kernel as $K_t = K_{t,v} * K_{t,h}$. They do not report any loss of quality compared to the use of 2D kernels, while smaller amount of memory makes it possible to handle larger motions with larger N and interpolate frames with higher resolution in one pass.

Pair of 1D kernels for each pixel for both of the input frames is obtained using a convolutional neural network that takes frames $\mathbf{I}_0$ and $\mathbf{I}_1$ as input. Separable convolution is implemented as a layer of the network. In addition to the traditional loss function based on ℓ_1 norm, they also use perceptual loss $\mathcal{L}_F$ – a feature based loss function – for training. They empirically choose to use `relu4_4` layer of the VGG19 network [32] as the feature extractor ϕ.

$$\mathcal{L}_F = ||\phi(\mathbf{I}_{0.5}) - \phi(\widehat{\mathbf{I}_{0.5}})||_2^2 \tag{3.6}$$

3.3.3 Deep Video Frame Interpolation using Cyclic Frame Generation

The state-of-the-art method is proposed by Liu et al. in [31] and is based on the deep voxel flow method. Deep voxel flow [23] uses an end-to-end trainable deep convolutional network that contains a voxel flow layer – *"a per-pixel, 3D optical flow vector across space and time"* [23] that is used to synthesise the interpolated image from the input ones. An advantage of this method is that voxel flow is computed only as an intermediate layer, and it is not compared to optical flow ground truth, which is difficult to obtain.

In [31] they improve voxel flow method with the following three components to achieve state-of-the-art.

3.3.3.1 Cycle Consistency Loss

Comparing images with traditional functions such as ℓ_1 norm and minimising loss functions built on them often leads to blurry results. They propose a solution to use already interpolated frames as input to the model again and predict the original input frame. More precisely, model f is used to predict 4 frames (Figure 3.2) during one run: $\mathbf{I}_1^{'} = f(\mathbf{I}_0, \mathbf{I}_2)$, $\mathbf{I}_1^{''} = f(\mathbf{I}_{0.5}^{'}, \mathbf{I}_{1.5}^{'})$, $\mathbf{I}_{0.5}^{'} = f(\mathbf{I}_0, \mathbf{I}_1)$ and $\mathbf{I}_{1.5}^{'} = f(\mathbf{I}_1, \mathbf{I}_2)$. Loss function $\mathcal{L}$ over a batch of N triplets is than calculated as follows:

$$\mathcal{L} = \mathcal{L}_r + \mathcal{L}_c = \sum_{n=1}^{N} ||\mathbf{I}_{n,1}^{'} - \mathbf{I}_{n,1}||_1 + ||\mathbf{I}_{n,1}^{''} - \mathbf{I}_{n,1}||_1. \tag{3.7}$$

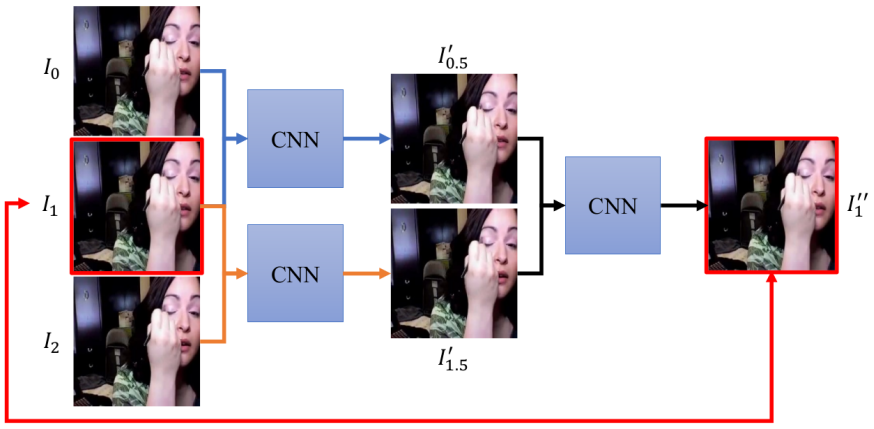


Figure 3.2: Diagram of cycle consistency loss. [31]

3.3.3.2 Motion Linearity Loss

They assume that *"time interval between two consecutive frames is short enough so that the motion is linear between the two frames"* [31]. Based on this assumption, the flow map $\mathbf{F}_{0\to 2}$ between $\mathbf{I}_0$ and $\mathbf{I}_2$ should be twice as large as the flow map $\mathbf{F}_{0.5\to 1.5}$ between $\widehat{\mathbf{I}_{0.5}}$ and $\widehat{\mathbf{I}_{1.5}}$. Motion linearity loss $\mathcal{L}_m$ is then used as a regularization parameter for optical flow estimation:

$$\mathcal{L}_m = \sum_{n=1}^{N} ||\mathbf{F}_{n,0\to 2} - 2 \cdot \mathbf{F}_{n,0.5\to 1.5}||_2^2. \tag{3.8}$$

3.3.3.3 Edge-guided Training

They have conducted experiments to show that interpolation of regions with high number of edges is difficult. To address this problem, they compute edge maps of input images using the HED convolutional network [33]. Edge map is added to the input to improve interpolation by preserving edge structure.

3.4 Related Work to Image Sequence Extrapolation

Two different approaches to the image sequence extrapolation using neural networks are examined in this section. Villegas et al. in [34] propose a method based on a motion of high-level features and prediction using a recurrent neural network. A method that is currently achieving state-of-the-art results is described by Reda et al. in [24].

3.4.1 Learning to Generate Long-term Future via Hierarchical Prediction

Villegas et al. [34] are describing a method for long-term future prediction. The majority of frame extrapolation methods are predicting future frames in a pixel-to-pixel manner. While these approaches can produce sharp results for few time steps ahead, predicted frames contain some pixel-level noise. These errors tend to exponentially amplify through time until they overwhelm the video signal.

In [34], authors are predicting human movement in front of a static camera. They propose a model for long-term frame prediction from high-level structures of previous frames. Human poses extracted from the images are used as these high-level structures in the paper. Prediction of the future poses is done by a LSTM [35] network. The network takes a sequence of high-level structures $\mathbf{p}_{1:t}$ as input and produces a high-level structure sequence $\mathbf{p}_{t+1:t+T}$ on the output.

Actual future frames are synthesized from predicted structures $\mathbf{p}_{t+1:t+T}$ and last observed frame x_t according to the diagram in Figure 3.3. The synthesis follows an analogy that structure $\mathbf{p}_t$ is to $\mathbf{p}_{t+n}$ as image x_t is to x_{t+n}. Their image generator uses two convolutional encoders f_{img} and f_{pose} to map images and high-level structures respectively to a feature space, where the pose feature transformations can be applied to synthesize features of the feature frame. The frame is afterwards decoded using convolutional decoder f_{dec}.

3.4.2 SDC-Net: Video prediction using spatially-displaced convolution

Reda et al. propose in [24] method for frame sequence extrapolation that is achieving state-of-the-art in both qualitative and quantitative evaluation and is inspired by the use of adaptive separable convolution in [22]. They introduce *Spatially Displaced Convolution* (SDC) which combines the adaptive convolution for each pixel with frequently used motion vectors. The pixel value in position (x, y) of first predicted frame $\mathbf{I}_{t+1}$ is computed as follows:

$$\mathbf{I}_{t+1}(x, y) = K(x, y) * \mathbf{P}_t(x + u, y + v), \tag{3.9}$$

where $K(x, y) \in R^{N \times N}$ is a 2D convolution kernel, (u, v) is a motion vector and $\mathbf{P}_t(x + u, y + v)$ is a patch of size $N \times N$ from the last observed image $\mathbf{I}_t$ with the center at $(x + u, y + v)$. Thanks to the motion vectors they can handle large motion and obtain the visually pleasing results from the spatially-adaptive convolution, despite the relatively small $N = 11$. To further reduce memory usage, they adopt separable convolutions from [22] where the 2D

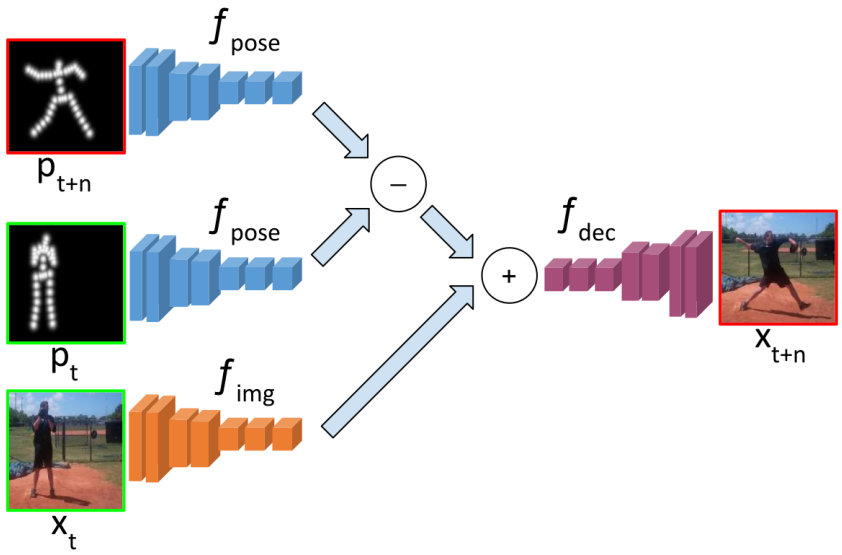


Figure 3.3: Diagram of image generator. [34]

kernel $K(x, y)$ is estimated with the pair of 1D kernels as showed in Section 3.3.2.

The prediction model is formulated as:

$$\mathbf{I}_{t+1} = \mathcal{T}(\mathcal{G}(\mathbf{I}_{1:t}, \mathbf{F}_{2:t}), \mathbf{I}_t). \tag{3.10}$$

$\mathcal{T}$ denotes the transformation process realized with SDC and described in equation 3.9. $\mathcal{G}$ is a fully convolutional network estimating parameters for $\mathcal{T}$. The network takes as input a sequence of t images $\mathbf{I}_{1:t}$ and a sequence of $t-1$ optical flows $\mathbf{F}_{2:t}$. $\mathbf{F}_i$ is defined as bacward optical flow between images $\mathbf{I}_i$ and $\mathbf{I}_{i-1}$. It should be noted that due to occlusion, predicted motion vectors (u, v) are not the same as optical flow $\mathbf{F}_{i+1}$.

Chapter 4

Dataset

This chapter elaborates on the process of creating a dataset for training and testing machine learning models for weather radar sequence interpolation and extrapolation.

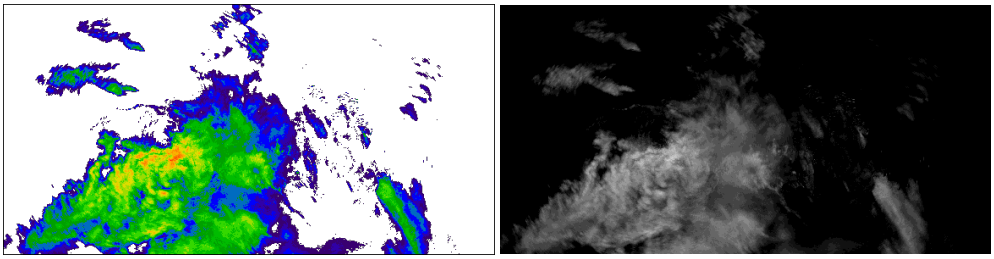
4.1 Source Images

I have chosen radar images from the area above the Czech Republic as a source for the dataset. The data were provided by the company Meteopress from the programme OPERA (Section 1.2). Images are composites from multiple weather radar networks, primary from the CZRAD network (Section 1.1.1.1), but also neighbouring radar networks may be included. The intensity of radar echoes is displayed in 16 discrete values from 0 to 60 dBZ (metric for intensity of the radar echo), coded by the colour space in Figure 4.2. 43776 images from the time frame from 1. 1. 2018 to 31. 10. 2018 were used.

Images are saved in the `.png` file format, in RGB colourspace with dimensions $[580 \times 294]$ (Figure 4.1a). The intensity of a radar echo is only one dimensional, so there is no need to work with RGB images as they occupy more memory than grayscale (Figure 4.1b) but do not give any more information (on the other hand, they are way better in communicating information to humans). To reduce the number of channels, colour of every pixel was matched with the colourspace in Figure 4.2 and replaced with the number $i \times 17$, where $i \in \{0, 1, \ldots, 15\}$ is the index of intensity from 0 to 15.

4.2 Data Augmentation

Due to computational limitations, I decided to reduce the image dimensions to $[96 \times 96]$. The fact that factorization of $96 = 2^5 \times 3$ is important as used CNNs



(a) RGB (b) grayscale

Figure 4.1: [580 × 294] weather radar image above the Czech Republic.

Figure 4.2: Colour coding of radar echo intensity.

contain pooling layers (Section 2.1.2) with $(2, 2)$ filters and $(2, 2)$ strides. This work should serve rather as a test of concept than as a final solution and therefore [96 × 96] format is fully adequate.

From each of the original images, 55 patches of size [96 × 96] were cropped, starting in the left top corner and sliding across the image with a step of size 48 in both of the directions. All of the patches from one position form a new sequence of 43776 frames and are processed separately. Each sequence is rotated $(i \bmod 4) * 90°$ clockwise, where $i \in \{0, 1, \ldots, 54\}$ is index of the sequence, to reduce biasing. Each place on the Earth has it owns atmosphere, and wind there blows more in some directions than the others. This operation aims to make the dataset balanced in precipitation motion for each direction.

4.3 Dataset for Interpolation Task

The dataset $(\mathbf{X}, \mathbf{y}); \mathbf{X}, \mathbf{y} \in \mathbb{R}^{N_{samples} \times N_{channels} \times 96 \times 96}$ has two parts, where $\mathbf{X}$ contains inputs and $\mathbf{y}$ corresponding ground truth. Images without or with a little precipitation have no valuable information for the dataset. Therefore, frames with more than 95% area without a radar echo, or where the strongest echo has only the first intensity, were removed. Each triplet of consecutive images $(\mathbf{I}_1, \mathbf{I}_2, \mathbf{I}_3)$ left in the sequence was split into the input and ground truth. Input images of one sample were concatenated along the second dimension, so they form one image with the separate ones as colour channels.

Algorithm `train_test_split()` from the library scikit-learn[3] was used to

[3] `https://scikit-learn.org/stable/`

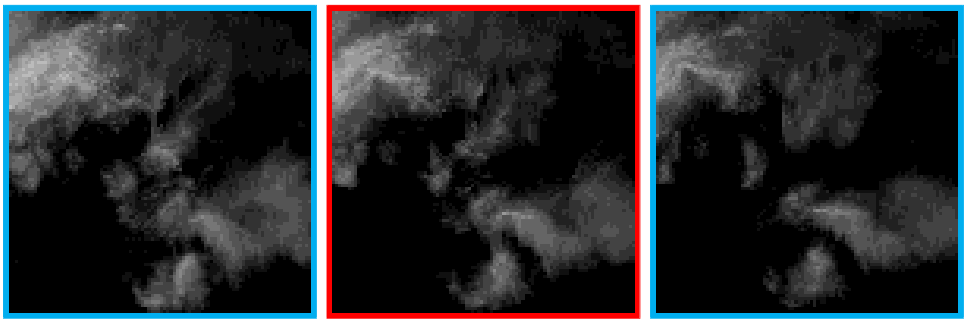


Figure 4.3: Example from the interpolation dataset. Blue framed images are the input ones and the image in the red frame is the ground truth for the output.

split the dataset into training, validation and test set with shapes as in Table 4.1. An example from the dataset is showed in Figure 4.3.

Table 4.1: Shapes of the datasets for the interpolation task.

set	**X**	**y**
train	$[142101 \times 2\times, 96 \times 96]$	$[142101 \times 1 \times 96 \times 96]$
validation	$[35526 \times 2 \times 96 \times 96]$	$[35526 \times 1 \times 96 \times 96]$
test	$[44407 \times 2 \times 96 \times 96]$	$[44407 \times 1 \times 96 \times 96]$

4.4 Dataset for Extrapolation Task

As the dataset for the interpolation task, the one for the extrapolation task (Section 3.1.2) has form $(\mathbf{X}, \mathbf{y}); \mathbf{X}, \mathbf{y} \in \mathbb{R}^{N_{samples} \times 3 \times 96 \times 96}$. Each sample consists of a sequence of 6 images $(\mathbf{I}_1, \mathbf{I}_2, \ldots, \mathbf{I}_6)$. The sequence $(\mathbf{I}_1, \mathbf{I}_2, \mathbf{I}_3)$ serves as an input, while the rest $(\mathbf{I}_4, \mathbf{I}_5, \mathbf{I}_6)$ as a ground truth sequence. Rest of the process is analogic to the Section 4.3. Both of the sequences for each sample are concatenated along the second dimension. Shapes of the datasets after splitting to the training, validation and test set are in the Table 4.2. An example from the dataset can be seen in Figure 4.4.

4.5 Implementation Details

Images are originally saved as `.png` files and it is worked with them as with numpy arrays during the whole process. Functions `imread()` and `imwrite()` from the library OpenCV[4] were used to load them into numpy array and again

[4] `https://opencv.org/`

Table 4.2: Shapes of the datasets for the extrapolation task.

set	**X**	**y**
train	$[67613 \times 3 \times 96 \times 96]$	$[67613 \times 3 \times 96 \times 96]$
validation	$[16904 \times 3 \times 96 \times 96]$	$[16904 \times 3 \times 96 \times 96]$
test	$[21130 \times 3 \times 96 \times 96]$	$[21130 \times 3 \times 96 \times 96]$

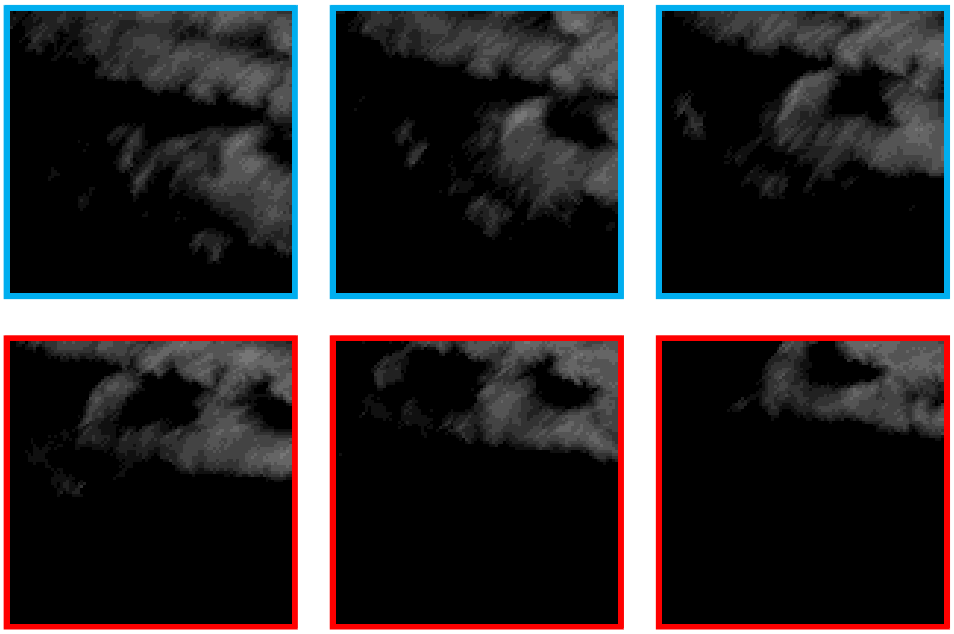


Figure 4.4: Example from the extrapolation dataset. Blue framed images form the input sequence and the images in the red frame are the ground truth for the output sequence.

save as `.png`. The final datasets are saved as numpy `.npy` files. The code for individual operations can be found in `tools._dataset_tools` python module and the whole process of dataset creation in `dataset.ipynb` jupyter notebook.

Chapter 5

Image Sequence Interpolation

I propose and describe a method for weather radar image sequence interpolation in this chapter.

5.1 Architecture

I have created a deep convolutional neural network (CNN), inspired by the work in [21, 22]. As described in Section 4.3, input to the network is a batch of 3D tensors with dimensions $[2 \times 96 \times 96]$. Convolutional layers consider the information through the whole depth (number of channels) of the input tensor, so adding more channels is one of the ways, how to input more information to a CNN. The idea behind the network is that convolutional layers will learn to extract high-level features from the input, describing the precipitation in the middle of the motion between the two frames. Afterwards, the interpolated image is synthesized using further convolutional and upsampling layers.

The overview of the CNN architecture can be seen in Figure 5.1. The basis of the architecture are convolutional blocks – stacks of convolutional layers. Each block consists of three convolutional layers with the kernel of size $(3, 3)$ and stride $(1, 1)$. [32] indicates that using a stack of convolutional layers has multiple benefits. The receptive field (Section 2.1.1.2) of the convolutional block is $(7, 7)$. Bigger receptive field means in this case that the network can better capture bigger motion in the input (pooling layers also work to the benefit of this). A single convolutional layer with $(7, 7)$ kernel has the same receptive field and $7^2C^2 + 1 = 49C^2 + 1$ trainable parameters, where C is the depth of both the input and the output and 1 is for bias. For comparison, the stack has only $3 \cdot (3^2C^2 + 1) = 27C^2 + 3$ trainable parameters, which should help with the regularisation, while decreasing the memory size of the network. Moreover, the activation function will be applied multiple times,

which can help with modelling of the nonlinearity. The exact output shapes of the convolutional blocks are displayed in Table 5.1. Input shape is $[2 \times 96 \times 96]$ and output of the last convolutional block is also the output of the network.

Table 5.1: Output shapes of convolutional blocks.

	conv32	conv64	conv128	conv256	conv256_256	deconv128	deconv64	deconv32	deconv1
depth	32	64	128	256	256	128	64	32	1
width	96	48	24	12	6	12	24	48	96
height	96	48	24	12	6	12	24	48	96

As activation function I have chosen PReLU (Section 2.3.1). The choice is based on the ReLU activation, which is frequently used in CNNs, has low computational complexity and is fast learning. The PReLU has an added advantage of non-zero gradient even in case of negative input. It is used as activation after each convolutional layer except the output one that has linear activation.

During the downscaling, the convolutional blocks are followed by the average-pooling layers with $(2, 2)$ kernels to reduce dimensionality and therefore extract only the important information. Upscaling is performed by the upsampling blocks consisting of an upsample layer and single convolutional layer with the same parameters as in the convolutional blocks. The upsampling has a scale factor of 2 and uses bilinear interpolation. The residual connections (Section 2.3.2) are included to improve fine-grained details of the output, which are essential if the produced images are to be visually pleasing.

Last but not least, batch normalisation (Section 2.3.3) is inserted in the convolutional blocks before the last activation function to reduce overfitting.

5.2 Loss Function

I have experimented with multiple loss functions (Section 2.2) that were minimalised during the training of the network. In addition to the traditional $\mathcal{L}_1$ loss (it is reported in [22] that it leads to better results than $\mathcal{L}_{MSE}$ loss when comparing images), I have tried a perceptual loss.

5.2.1 VGG19 Loss

VGG19 net [32] is a deep convolutional network that achieved state-of-the-art results on ILSVRC-14 (1^{st} in localisation and 2^{nd} in classification task on ImageNet dataset [36]). In the paper, they concentrate on the effect of stacking convolutional layers with smaller kernel sizes the same way as I do in this work. The VGG19 (alongside with other state-of-the-art neural networks) can be found pretrained in the pytorch library, which makes it easy to use it

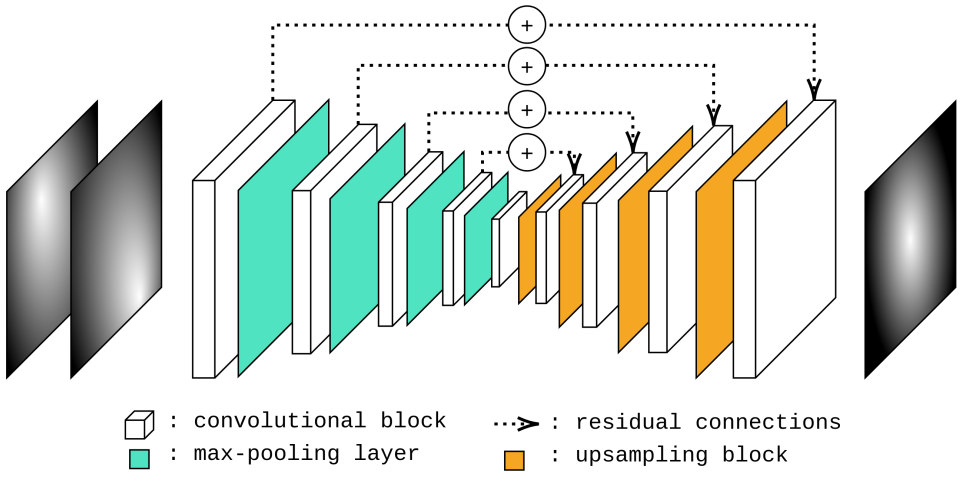


Figure 5.1: The architecture of the CNN.

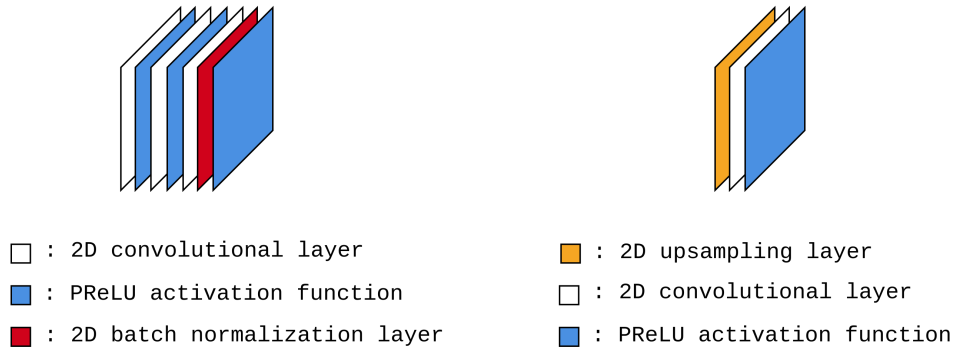


Figure 5.2: The architecture of the convolutional and upsample block.

as a feature extractor. As the vector of extracted features $\mathcal{F}$ of the input image $\mathbf{I}$ is taken the output $\phi_c(\cdot)$ of some convolutional layer c of the network

$$\mathcal{F}(\mathbf{I}) = \phi_c(\mathbf{I}). \tag{5.1}$$

The actual loss function $\mathcal{L}_F$ between ground truth image $\mathbf{I}$ and inferred image $\widehat{\mathbf{I}}$ is than computed similarly to the feature reconstruction loss in [25]:

$$\mathcal{L}_F(\mathbf{I}, \widehat{\mathbf{I}}) = ||\phi_c(\mathbf{I}) - \phi_c(\widehat{\mathbf{I}})||_2. \tag{5.2}$$

One small issue is that the VGG19 expects the input in the RGB colourspace. To solve this problem, both of the images are concatenated three times along the channel dimension.

ConvNet Configuration						
A	A-LRN	B	C	D	E	
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers	
input (224 × 224 RGB image)						
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64	
maxpool						
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	`relu2_2`
maxpool						
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 **conv3-256**	`relu3_4`
maxpool						
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 **conv3-512**	`relu4_4`
maxpool						
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 **conv3-512**	`relu5_4`
maxpool						
FC-4096						
FC-4096						
FC-1000						
soft-max						

Figure 5.3: Architectures of the different VGG convolutional neural networks [32]. Considered output layers of the VGG19 network are highlited red.

I have considered 4 different output layers of the VGG19 (Figure 5.3) as c for $\mathcal{L}_F$. With each variant of the loss function my network was trained for 5 epochs (after training for 50 epochs with $\mathcal{L}_1$) and evaluated on the validation dataset with SSIM index. The results can be seen in Table 5.2 and examples in Figure 5.4. Based on these results, I have chosen the $c =$ `relu2_2` for $\mathcal{L}_F$, as it outperforms the other options and results look empirically the most simillar to the ground truth. From now on the $\mathcal{L}_F$ will be considered with $c =$ `relu2_2`.

Table 5.2: Values of average SSIM index on the validation dataset for various $\mathcal{L}_F$ loss functions.

Loss function	**SSIM**
$\mathcal{L}_F$ with $c =$ `relu2_2`	0.8423
$\mathcal{L}_F$ with $c =$ `relu3_4`	0.7679
$\mathcal{L}_F$ with $c =$ `relu4_4`	0.7650
$\mathcal{L}_F$ with $c =$ `relu5_4`	0.8068

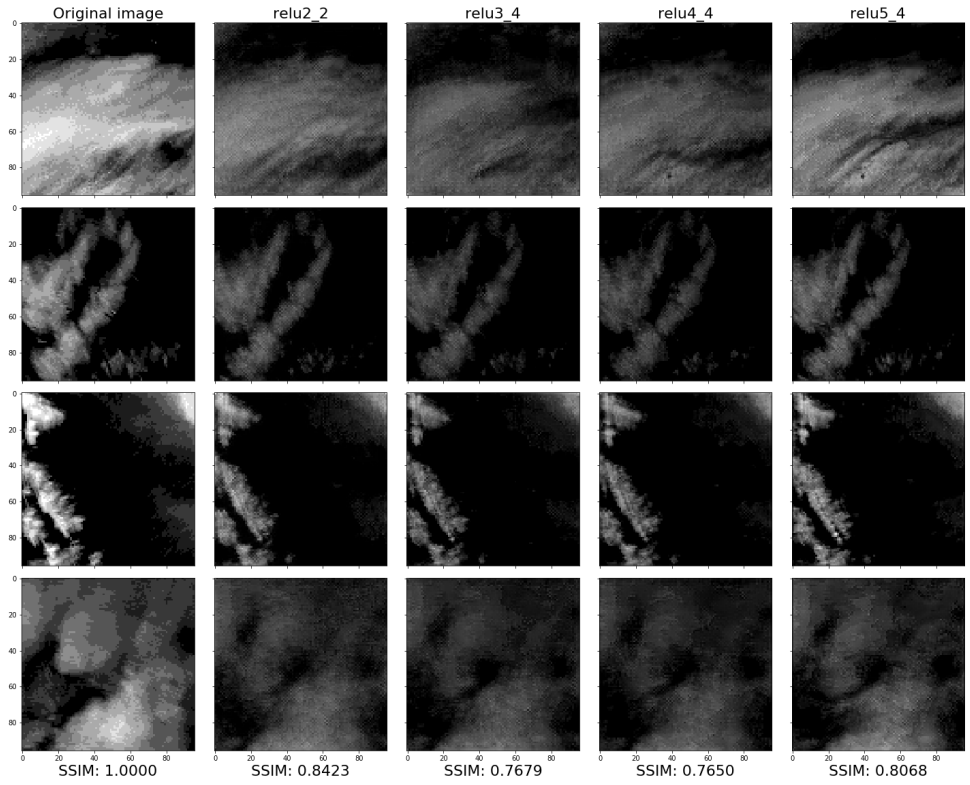


Figure 5.4: Examples and average SSIM of the validation dataset after training with different variants of $\mathcal{L}_F$.

5.2.2 Combined Loss

Combined loss $\mathcal{L}_C$ is a combination of $\mathcal{L}_1$ and $\mathcal{L}_F$

$$\mathcal{L}_C(\mathbf{I}, \widehat{\mathbf{I}}) = \mathcal{L}_1(\mathbf{I}, \widehat{\mathbf{I}}) + \alpha \cdot \mathcal{L}_F(\mathbf{I}, \widehat{\mathbf{I}}), \tag{5.3}$$

where α is a parameter to balance the effect of the two loss functions.

Values $1{\cdot}10^{-5}$, $2{\cdot}10^{-5}$ and $3{\cdot}10^{-5}$ were tried for the parameter α. Analogically to the previous section the network was trained for 5 epochs with these loss functions and afterwards evaluated using the SSIM index on the validation dataset. Table 5.3 contains the results (pure $\mathcal{L}_F$ is included for comparison) of validation and examples are displayed in Figure 5.5. The value $\alpha = 1 \cdot 10^{-5}$ closely wins this comparison, while empirically looking slightly better too. It will be referred to $\mathcal{L}_C$ as with parameter $\alpha = 1 \cdot 10^{-5}$.

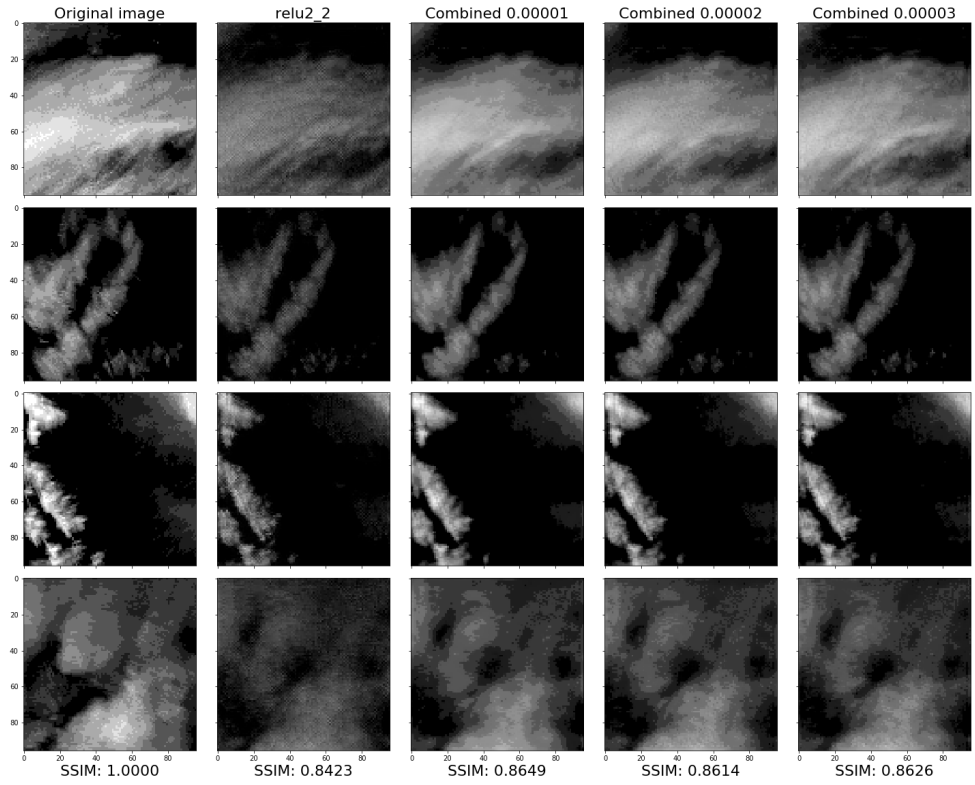


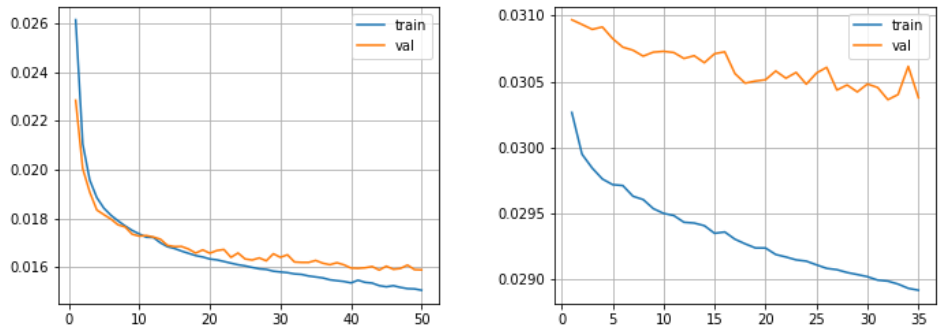
Figure 5.5: Examples and average SSIM of the validation dataset after training with different variants of $\mathcal{L}_C$.

Table 5.3: Values of average SSIM index on the validation dataset for various $\mathcal{L}_C$ loss functions.

Loss function	**SSIM**
$\mathcal{L}_C$ with $\alpha = 1 \cdot 10^{-5}$	0.8649
$\mathcal{L}_C$ with $\alpha = 2 \cdot 10^{-5}$	0.8614
$\mathcal{L}_C$ with $\alpha = 3 \cdot 10^{-5}$	0.8626
$\mathcal{L}_F$	0.8423

5.3 Training

The network was trained on the created dataset for the interpolation task. The training was started with the use of $\mathcal{L}_1$ loss function (Figure 5.6a). I stopped the training after 50 epochs because the value of validation loss stopped improving even tough the training loss was still decreasing. I interpret this that



(a) $\mathcal{L}_1$ (b) $\mathcal{L}_C$

Figure 5.6: Plot of loss on the training and validation data sets during the training.

the network stopped learning the problem at this point and started to learn the exact examples in the dataset.

An output of the network after 50^{th} epoch is displayed in Figure 5.7. It can be noticed that the network can capture well the shape of the radar echo, but it lacks the internal structure of radar echo (best visible in the top right corner). This problem is addressed with further training guided by the combined loss $\mathcal{L}_C$ which has the exact form:

$$\mathcal{L}_C(\mathbf{I}, \widehat{\mathbf{I}}) = \mathcal{L}_1(\mathbf{I}, \widehat{\mathbf{I}}) + 1 \cdot 10^{-5} \cdot ||\phi_{\text{relu2\_2}}(\mathbf{I}) - \phi_{\text{relu2\_2}}(\widehat{\mathbf{I}})||_2 \tag{5.4}$$

The training with $\mathcal{L}_C$ went on for 35 epochs. Even though the validation loss may seem slightly decreasing, as indicated in Figure 5.6b, the improvement over the last 15 epochs is empirically hardly noticeable, which is the reason for finally stopping training after 85 epochs.

5.4 Results

The final trained network was tested on the test part of the dataset using the SSIM index. It is compared to a simple Euclidean method, which will be called mean. Using this method, each pixel of the output image is calculated as a mean of the input pixels at that position. The performance of the final network is not tested against a method estimating interpolated images based on the optical flow (Section 3.2), due to a lack of suitable implementation of the mentioned algorithm.

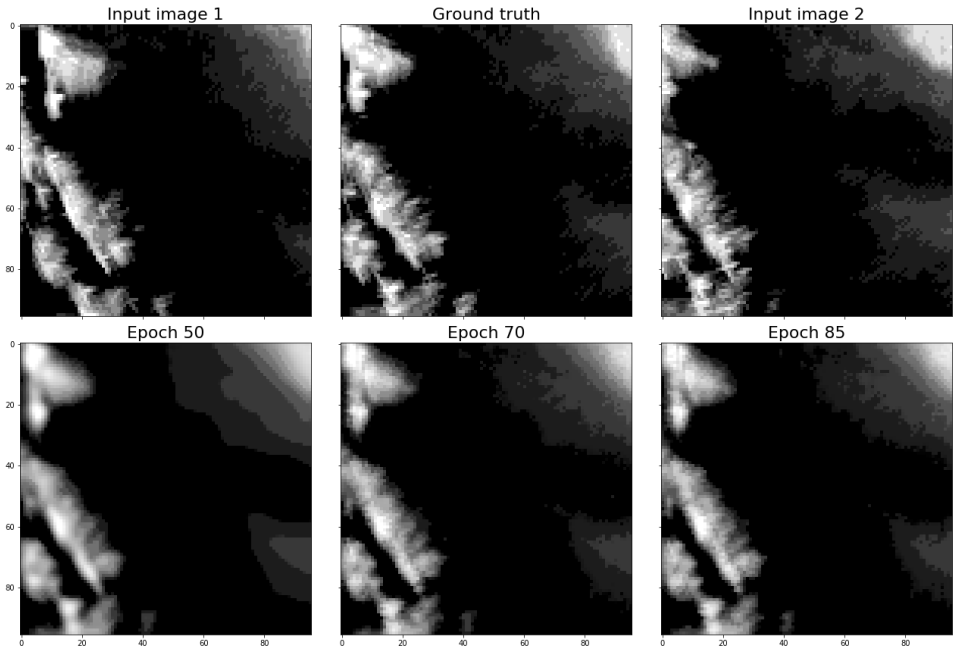


Figure 5.7: Output of the network after 50^{th}, 70^{th} and 85^{th} epoch.

Results of the testing are displayed in Table 5.4. The table shows that my network achieves significantly better results compared to the mean method.

Table 5.4: Values of average SSIM index on the test dataset for the interpolation task.

Method	**SSIM**
Final network	0.8646
Mean	0.7469

Empirical evaluation can be done on examples displayed in Figure 5.8. In terms of radar echo intensity, both methods perform very comparably to the ground truth. The mean model looks good at places with small motion but theoretically cannot handle any motion which can be seen at first of the examples, where radar echo at the new position has no overlap with the old position at all. On the other hand, the neural network handles this motion well and keeps the radar echo compact. The internal structure of the neural network interpolated radar echo is less eater than the ground truth, but areas where the precipitation intensity changes are distinguishable. In the mean method, the outputs are noisy, which makes it hard to read the intensity of precipitation at the target position.

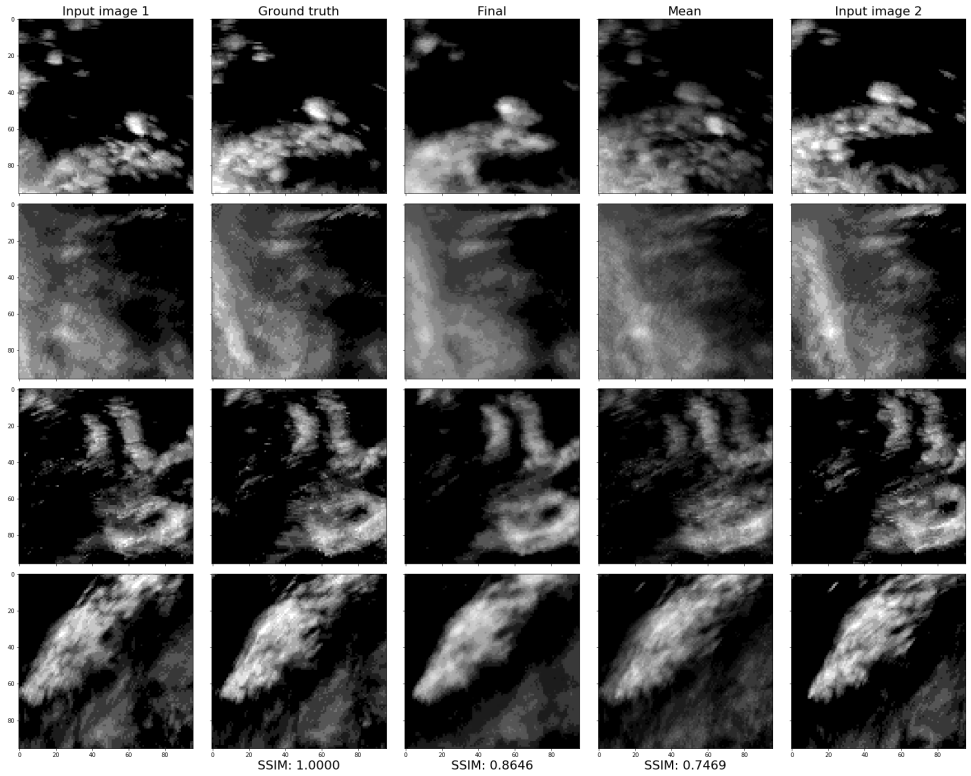


Figure 5.8: Examples of image sequence interpolation. From left to right: first input image, ground truth interpolated image, interpolated image from my network, mean method interpolated image, second input image.

5.5 Implementation Details

The convolutional neural network is implemented using the library PyTorch[5] and its code can be found in python module `models.interpolation`. All of the computations took place at the machine in the Google Compute Engine[6] with 26GB of RAM (Random Access Memory) and NVIDIA Tesla V100 GPU(Graphics Processing Unit). Training was done in the Jupyter Notebook `training_interpolation.ipynb` with the help of the class `Training` from the python module `tools._torch_tools`. The network was trained using the PyTorch implementation of the AdaMax[7] optimizer with default parameters (learning rate 0.002, $\beta_1 = 0.9$ and $\beta_2 = 0.999$). The size of a mini batch is 32 samples.

[5]https://pytorch.org/
[6]https://cloud.google.com/compute
[7]https://pytorch.org/docs/stable/optim.html?highlight=adamax#torch.optim.Adamax

The datasets are stored as a `.npy` files and loaded to RAM before the start of the training. The method `tools._torch_tools.Training.getBatch()` is in charge of preparing mini-batches. Each batch, when needed, is first taken from the numpy array and afterwards sent to the GPU memory. After each epoch the weights of the model were saved as `.pth` file with the `torch.save()` function.

Chapter 6

Image Sequence Extrapolation

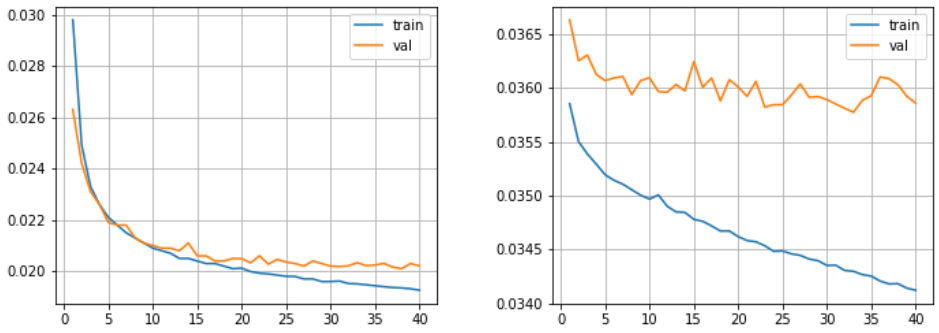
This chapter contains the details about the method for the extrapolation task. I have used the same network as in Chapter 5 with one minor change. The network takes three consecutive images $(\mathbf{I}_1, \mathbf{I}_2, \mathbf{I}_3)$ as the input instead of two and it is trained to predict one following image $\widehat{\mathbf{I}}_4$ after the three input ones.

6.1 Multiple-step Extrapolation

While the network is trained to predict one following image, in a real-world scenario, it is expected to have prediction further in the future than one time step. A solution to this is to take the output of the network $\widehat{\mathbf{I}}_4$ and put it in the input sequence $(\mathbf{I}_1, \mathbf{I}_2, \mathbf{I}_3, \widehat{\mathbf{I}}_4)$. Last three images of this sequence $(\mathbf{I}_2, \mathbf{I}_3, \widehat{\mathbf{I}}_4)$ are again fed into the network to obtain prediction $\widehat{\mathbf{I}}_5$ at next time step. Using this technique, it is possible to predict an arbitrary number of future images. On the other hand, any error introduced by the network will quickly scale up, because for the following time steps it is already incorporated in the input. In this thesis, I will work with a prediction for three steps into the future, so always at least one of the input images is ground truth. This algorithm is implemented in the method `models.extrapolation.Model().predict()`.

6.2 Training

The network was trained on the created dataset for the extrapolation task (Section 4.4), to predict the first image of the output sequence. $\mathcal{L}_1$ loss function was minimised for the first 40 epochs of the training and the progress is visible in Figure 6.1a. Example of the network output after the stabilisation of the validation error can be found in Figure 6.2. Similarly to the training of the network for the interpolation task (Section 5.3), the results after minimising $\mathcal{L}_1$ are blurry.



(a) $\mathcal{L}_1$ (b) $\mathcal{L}_C$

Figure 6.1: Plot of loss on the training and validation data sets during the training.

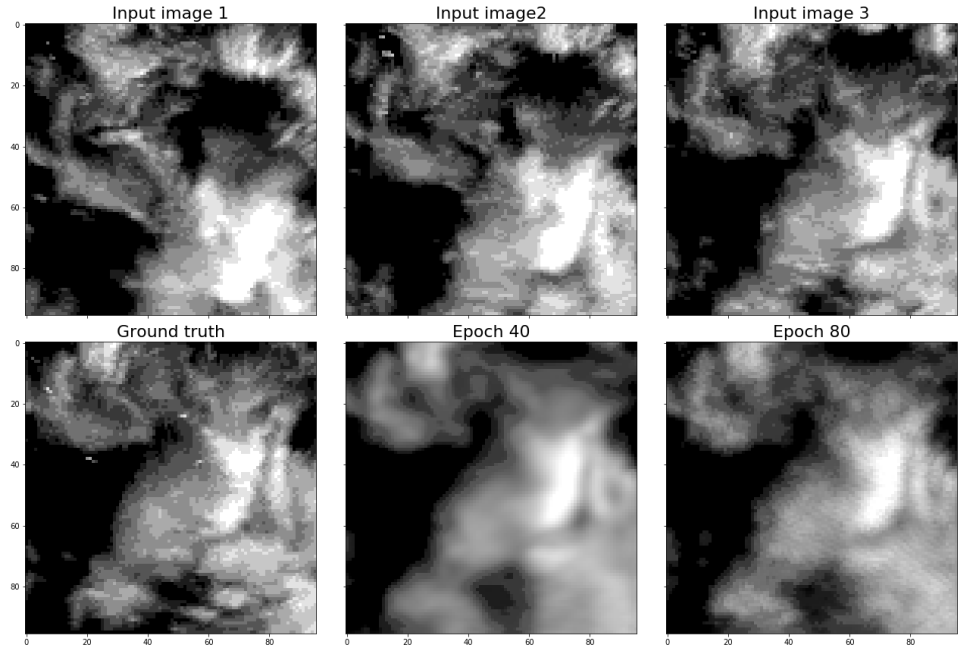


Figure 6.2: Output of the network after 40^{th} and 80^{th} epoch.

For the following 40 epochs the network was trained using the same $\mathcal{L}_C$ as in the Section 5.3. The plot of training and validation error (Figure 6.1b) indicates that the validation error stopped improving after 5^{th} epoch, and since then only oscillated around a constant value. The output of the network after finally stopping the training can be seen in Figure 6.2.

6.3 Results

The SSIM index was used as a metric for quantitative evaluation of the trained network. Results of the network are compared to the method COTREC described in Section 1.3.1.

Results of evaluation on the test part of the dataset are displayed in Table 6.1. Both of the methods were evaluated separately for each step of the prediction. It is not surprising that the SSIM index is decreasing as predicting frames further into the future.

My network outperformed the COTREC method. However, settings of this testing were unfavourable for the COTREC. COTREC is only shifting the signal it got on the input, which leads to black bars around the borders of the extrapolated image. In this setting, where the input is of shape $[96 \times 96]$ the black borders take up a massive portion of the image, which is naturally decreasing SSIM.

Table 6.1: Values of average SSIM index on the test dataset for the extrapolation task.

	SSIM in a step of prediction		
Method	**1st**	**2nd**	**3rd**
Final network	0.8383	0.7657	0.7162
COTREC	0.7663	0.6797	0.6295

Empirical observations can be done on the example in Figure 6.3. In the ground truth sequence, the shape of the radar echo is not only shifting but also changing its shape. COTREC extrapolation cannot capture this change at all, while the one from the network can, even though not 100%. In terms of the internal structure of the radar echo, images produced by COTREC seem more realistically as they are not changing it. The network ones are blurry, but empirically, nor of the methods can extrapolate well a local change of precipitation intensity.

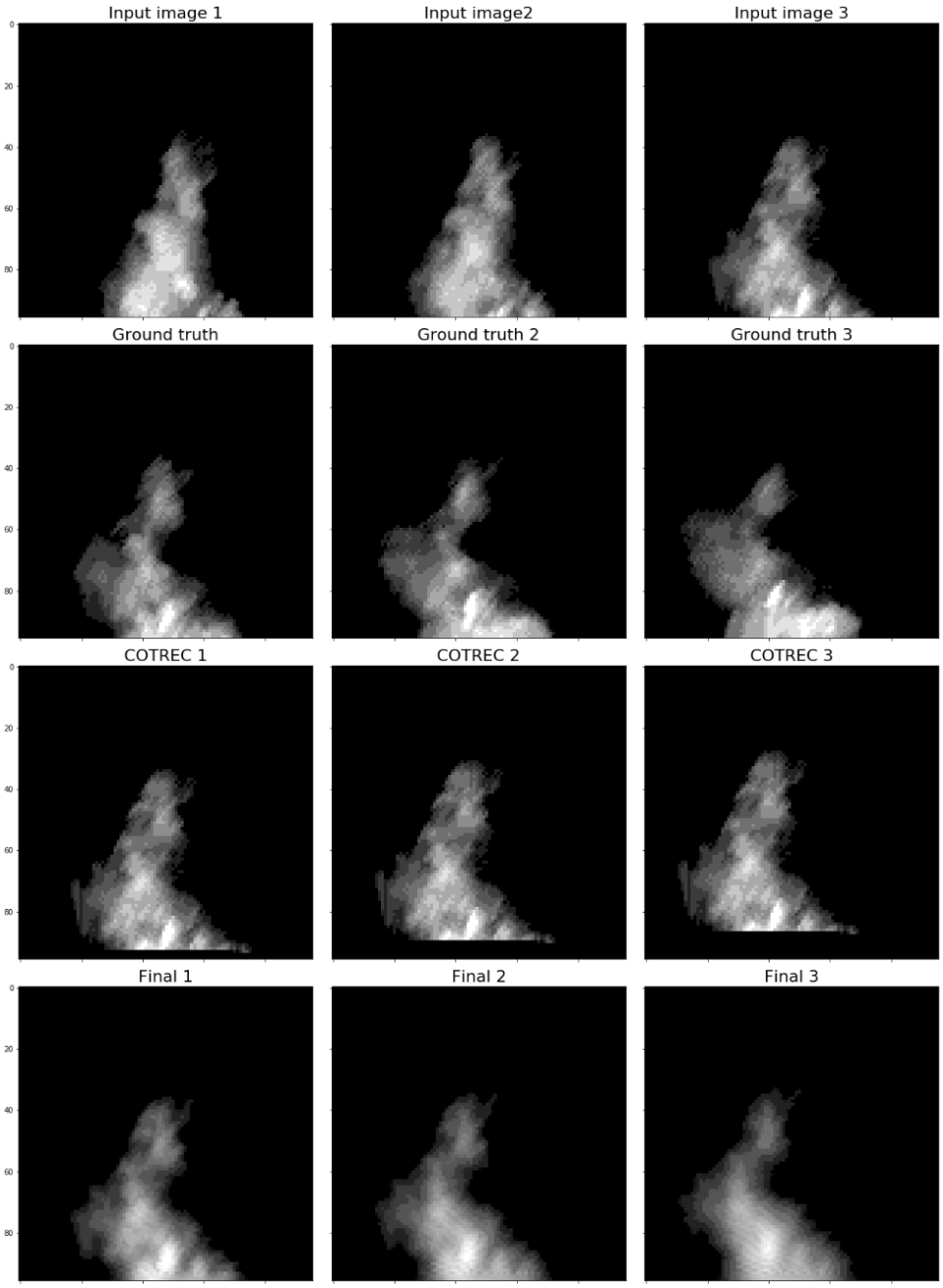


Figure 6.3: Example of extrapolation. Rows from top to bottom: input sequence, ground truth extrapolated sequence, COTREC extrapolated sequence, my network extrapolated sequence.

Conclusion

Due to recent development in communication, nowcasting as a communicating of short-term forecasts to the target user in real time is an up-to-date topic. In this thesis, I explored the possibilities of using machine learning algorithms for weather radar image sequence interpolation and extrapolation.

The objectives of the theoretical part of this thesis were to introduce the reader to the weather nowcasting, necessary machine learning concepts and research current machine learning methods for image sequence interpolation and extrapolation. I have completed these tasks respectively in Chapters 1, 2 and 3.

The first objective of the practical part was to create a dataset for both image sequence interpolation and extrapolation tasks. Furthermore, models for these tasks were to be designed, trained and evaluated, and a possible future work should have been outlined.

I describe the process of creating the dataset in Chapter 4. Later chapters show that the dataset is sufficient for designing and training of prototype machine learning models, even though motion between images in a sequence is often not very significant. I have created and trained convolutional neural networks for both of the tasks in Chapters 5 and 6.

I have compared the results of the proposed networks to the methods that do not use machine learning. Quantitative evaluation in both cases went in favour of my convolutional network. Although, it has to be noted that the method for interpolation was incomparably more simple than my network. Likewise, in the extrapolation task, the network was compared to the COTREC method, which is currently used, but the evaluation settings were unfavourable for the COTREC.

Based on the empirical evaluation of the results, I conclude that both of the networks achieve good looking results. In the extrapolation task, they are in multiple ways better than the currently used COTREC and in general promising.

Outline of Future Work

To the best of my knowledge, machine learning was used for the first time for the extrapolation of a sequence of weather radar images in the Czech Republic in this thesis. The results implicate that further research should be done in this field and that a nowcasting system containing these methods can be and should be created.

Future steps include adjusting the methods to the whole radar domain of the Czech Republic (instead of 96×96 input images) and to the domain of the whole Europe (this will be possible thanks to the composite images from the OPERA programme). Experimenting with both the architecture and the input data should be done to improve the performance of the methods, especially the quality of the internal structure of an extrapolated radar echo. Separable convolutions, recurrent neural networks or training the network to predict multiple time steps into the future directly will be tried. If successful, the results of this updated method will be published in real time and received a qualitative evaluation from the public will be used to decide if and how to continue in developing nowcasting systems.

Bibliography

1. MIGRATION IOM, International Organization for. *Migration and Climate Change* [online]. 2019 [visited on 2019-05-08]. Available from: https://www.iom.int/migration-and-climate-change-0.
2. PETTERSSEN, Sverre. *Introduction to meteorology*. Merz Press, 2007. ISBN 978-1406718225.
3. WORLD METEOROLOGICAL ORGANIZATION (WMO). *Guidelines for Nowcasting Techniques*. WMO, 2017. ISBN 978-92-63-11198-2.
4. MASS, Clifford; MASS, Clifford F, et al. Nowcasting: The next revolution in weather prediction. *Bulletin of the American Meteorological Society*. 2011, pp. 1–23.
5. WOLFF, Christian. *Radar Principle* [online]. 2019 [visited on 2019-04-15]. Available from: http://www.radartutorial.eu/01.basics/Physical%20fundamentals%20of%20the%20radar%20principle.en.html.
6. WOLFF, Christian. *Physical fundamentals of the radar principle* [online]. 2019 [visited on 2019-04-15]. Available from: http://www.radartutorial.eu/01.basics/Radar%20Principle.en.html.
7. KRČMÁŘ, Petr. Na návštěvě meteoradaru Skalky: kde se berou radarové mapy srážek. *Root.cz* [online]. 2019 [visited on 2019-04-18]. Available from: https://www.root.cz/clanky/na-navsteve-meteoradaru-skalky-kdyz-ctvrt-megawattu-sviti-do-mraku/.
8. HENDERSON, Tom. The Doppler Effect. *Physics Tutorial* [online]. 2019 [visited on 2019-05-08]. Available from: https://www.physicsclassroom.com/class/waves/Lesson-3/The-Doppler-Effect.

9. ČHMÚ. *Radarová síť ČHMÚ* [online]. 2018 [visited on 2019-04-15]. Available from: http://portal.chmi.cz/files/portal/docs/meteo/rad/info_czrad/index.html.

10. NAJMAN, Michal. Může malá firma vybudovat velkou radarovou síť? *Medium* [online]. 2018 [visited on 2019-04-15]. Available from: https://medium.com/pocasi/m%C5%AF%C5%BEe-mal%C3%A1-firma-vybudovat-velkou-radarovou-s%C3%AD%C5%A5-f3b721569b36.

11. ČHMÚ. *ČHMÚ – Nowcasting webportal* [online]. 2019 [visited on 2019-04-26]. Available from: http://portal.chmi.cz/files/portal/docs/meteo/rad/inca-cz/short.html.

12. EUMETNET, GIE. *OBSERVATIONS – OPERA* [online]. 2019 [visited on 2019-05-14]. Available from: http://eumetnet.eu/activities/observations-programme/current-activities/opera/.

13. MECKLENBURG, S; SCHMID, W; JOSS, J. COTREC-a simple and reliable method for nowcasting complex radar pattern over complex orography. In: *Final Seminar of COST-75*. 1999, pp. 441–450.

14. FROLÍK, Petr. *Využití extrapolace radarového echa pro kvantitativní předpověď srážek*. Univerzita Karlova, Matematicko-fyzikální fakulta, 2007. Diplomová práce.

15. LI, L; SCHMID, W; JOSS, J. Nowcasting of motion and growth of precipitation with radar over a complex orography. *Journal of applied meteorology*. 1995, vol. 34, no. 6, pp. 1286–1300.

16. NIELSEN, Michael A. *Neural networks and deep learning*. Determination press San Francisco, CA, USA: 2015.

17. KARPATHY, Andrej. *CS231n Convolutional Neural Networks for Visual Recognition* [online]. 2019 [visited on 2019-05-14]. Available from: http://cs231n.github.io/convolutional-networks/.

18. LONG, Jonathan; SHELHAMER, Evan; DARRELL, Trevor. Fully convolutional networks for semantic segmentation. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 3431–3440.

19. HIEN, Dang Ha The. A guide to receptive field arithmetic for Convolutional Neural Networks. *Medium* [online]. 2017 [visited on 2019-05-15]. Available from: https://medium.com/mlreview/a-guide-to-receptive-field-arithmetic-for-convolutional-neural-networks-e0f514068807.

20. TORCH CONTRIBUTORS. *torch.nn – PyTorch master documentation* [online]. 2018 [visited on 2019-05-14]. Available from: https://pytorch.org/docs/stable/nn.html.

21. LONG, Gucan; KNEIP, Laurent; ALVAREZ, Jose M; LI, Hongdong; ZHANG, Xiaohu; YU, Qifeng. Learning image matching by simply watching video. In: *European Conference on Computer Vision*. 2016, pp. 434–450.

22. NIKLAUS, Simon; MAI, Long; LIU, Feng. Video frame interpolation via adaptive separable convolution. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 261–270.

23. LIU, Ziwei; YEH, Raymond A; TANG, Xiaoou; LIU, Yiming; AGARWALA, Aseem. Video frame synthesis using deep voxel flow. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 4463–4471.

24. REDA, Fitsum A; LIU, Guilin; SHIH, Kevin J; KIRBY, Robert; BARKER, Jon; TARJAN, David; TAO, Andrew; CATANZARO, Bryan. SDC-Net: Video prediction using spatially-displaced convolution. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 718–733.

25. JOHNSON, Justin; ALAHI, Alexandre; FEI-FEI, Li. Perceptual losses for real-time style transfer and super-resolution. In: *European conference on computer vision*. 2016, pp. 694–711.

26. HE, Kaiming; ZHANG, Xiangyu; REN, Shaoqing; SUN, Jian. Deep residual learning for image recognition. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

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

28. WANG, Zhou; BOVIK, Alan C; SHEIKH, Hamid R; SIMONCELLI, Eero P, et al. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*. 2004, vol. 13, no. 4, pp. 600–612.

29. SCIKIT-IMAGE. *Structural similarity index* [online]. 2019 [visited on 2019-05-08]. Available from: `https://scikit-image.org/docs/dev/auto_examples/transform/plot_ssim.html`.

30. BAKER, Simon; SCHARSTEIN, Daniel; LEWIS, JP; ROTH, Stefan; BLACK, Michael J; SZELISKI, Richard. A database and evaluation methodology for optical flow. *International Journal of Computer Vision*. 2011, vol. 92, no. 1, pp. 1–31.

31. LIU, Yu-Lun; LIAO, Yi-Tung; LIN, Yen-Yu; CHUANG, Yung-Yu. Deep Video Frame Interpolation using Cyclic Frame Generation. In: *Thirty-Third AAAI Conference on Artificial Intelligence*. 2019.

32. SIMONYAN, Karen; ZISSERMAN, Andrew. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*. 2014.

33. XIE, Saining; TU, Zhuowen. Holistically-Nested Edge Detection. In: *The IEEE International Conference on Computer Vision (ICCV)*. 2015.

34. VILLEGAS, Ruben; YANG, Jimei; ZOU, Yuliang; SOHN, Sungryull; LIN, Xunyu; LEE, Honglak. Learning to generate long-term future via hierarchical prediction. In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. 2017, pp. 3560–3569.

35. OLAH, Christopher. Understanding LSTM Networks. *colah's blog* [online]. 2015 [visited on 2019-05-15]. Available from: `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`.

36. RUSSAKOVSKY, Olga et al. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*. 2015, vol. 115, no. 3, pp. 211–252. Available from DOI: `10.1007/s11263-015-0816-y`.

Appendix A

Acronyms

CHMI The Czech Hydrometeorological Institute

CNN Convolutional Neural Network

CZRAD The Czech Weather Radar Network

FCN Fully Convolutional Network

GPU Graphics Processing Unit

MSE Mean Squared Error

RAM Random Access Memory

SDC Spatially Displaced Convolution

SSIM Structural Similarity

Appendix B

Contents of enclosed DVD

- `readme.txt` ..... the file with DVD contents description
- `src` ..... the direcotry of source codes and data
  - `data` ..... the directory with weights of trained models
  - `examples` ..... the directory of example inputs
  - `models` ..... the directory with source codes of models
  - `tools` ..... the directory with python modules
  - `dataset.ipynb` ..... the notebook where dataset is created
  - `dataset_extrapolation.zip` ..... the extrapolation task dataset
  - `dataset_interpolation.zip` ..... the interpolation task dataset
  - `extrapolate.py` ..... python script for extrapolation
  - `interpolate.py` ..... python script for interpolation
  - `train_extrapolation.ipynb` ..... the notebook for training of the extrapolation model
  - `train_interpolation.ipynb` ..... the notebook for training of the interpolation model
- `text` ..... the thesis text directory
  - `fig` ..... the directory with figures
  - `bibliography.bib` ..... the bibliography resource
  - `thesis.pdf` ..... the thesis text in PDF format
  - `thesis.tex` ..... the LaTeX source code of the thesis

Appendix C

Examples of Extrapolation of Weather Radar Image Sequences

This appendix contains additional examples of interpolation with both my network and COTREC.

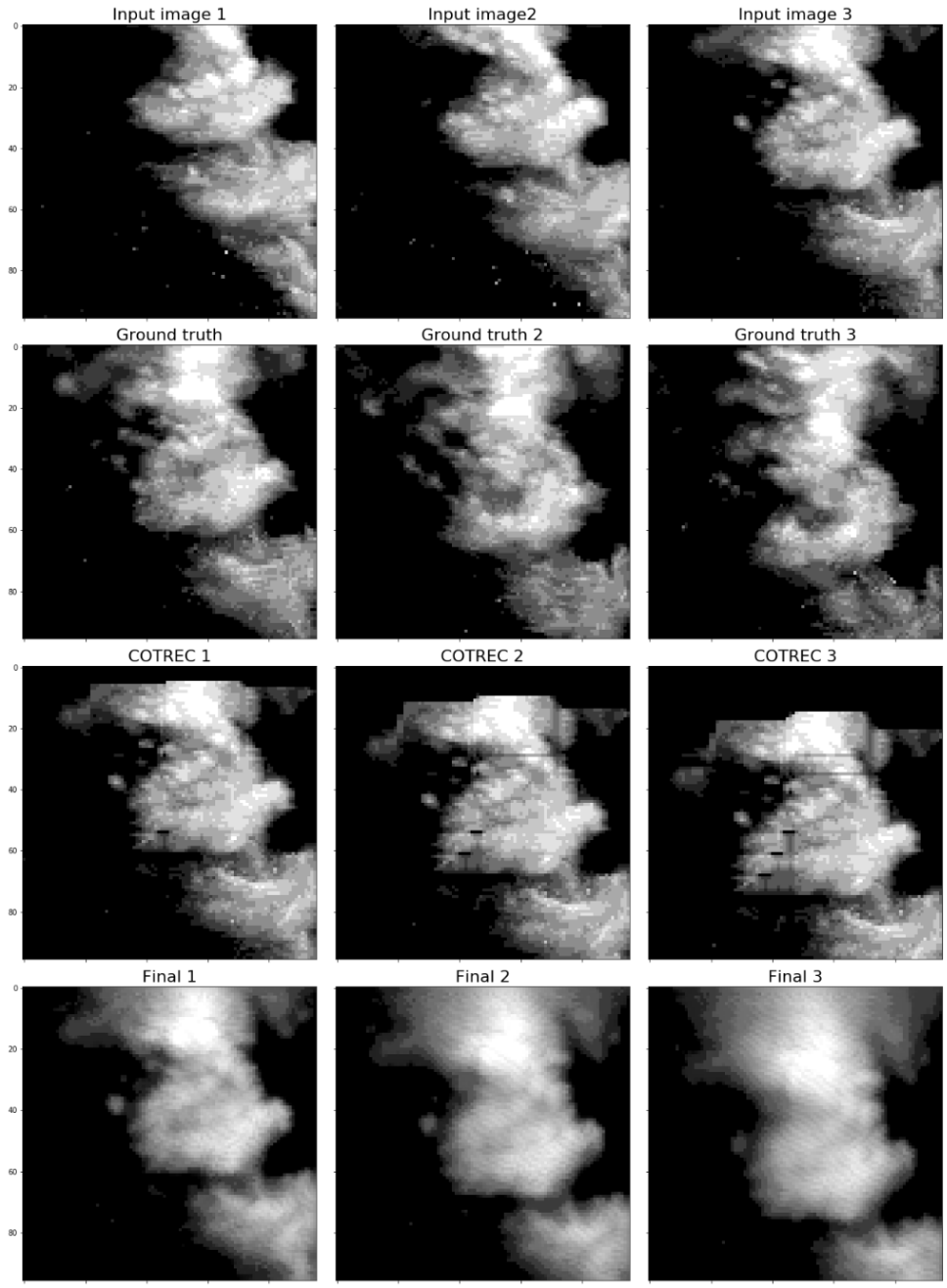


Figure C.1: It can be seen in this example how COTREC extrapolates the radar echo.

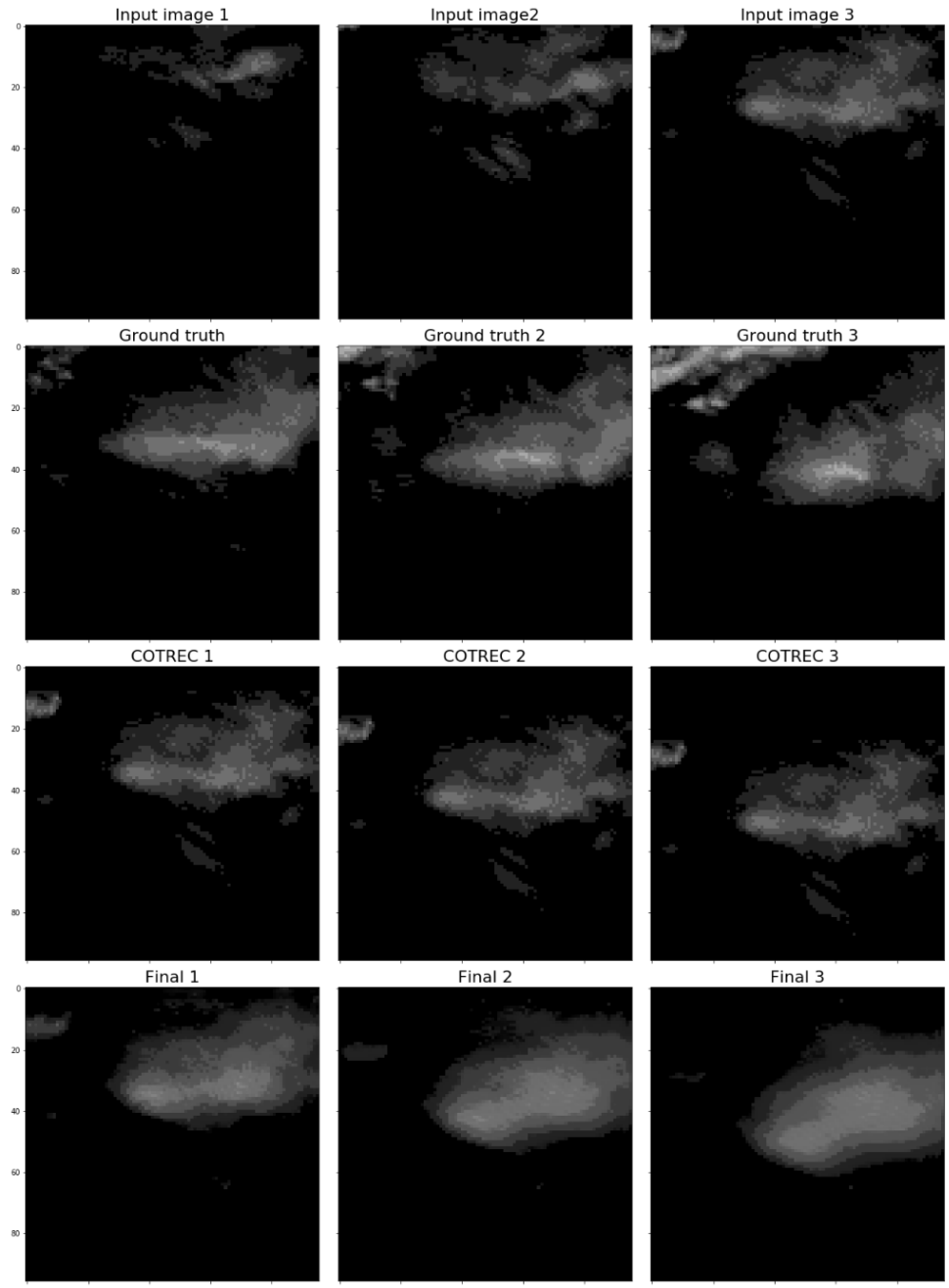


Figure C.2: COTREC has ideal conditions in this example, as the whole precipitation cell is in the frame and its intensity is not changing much.

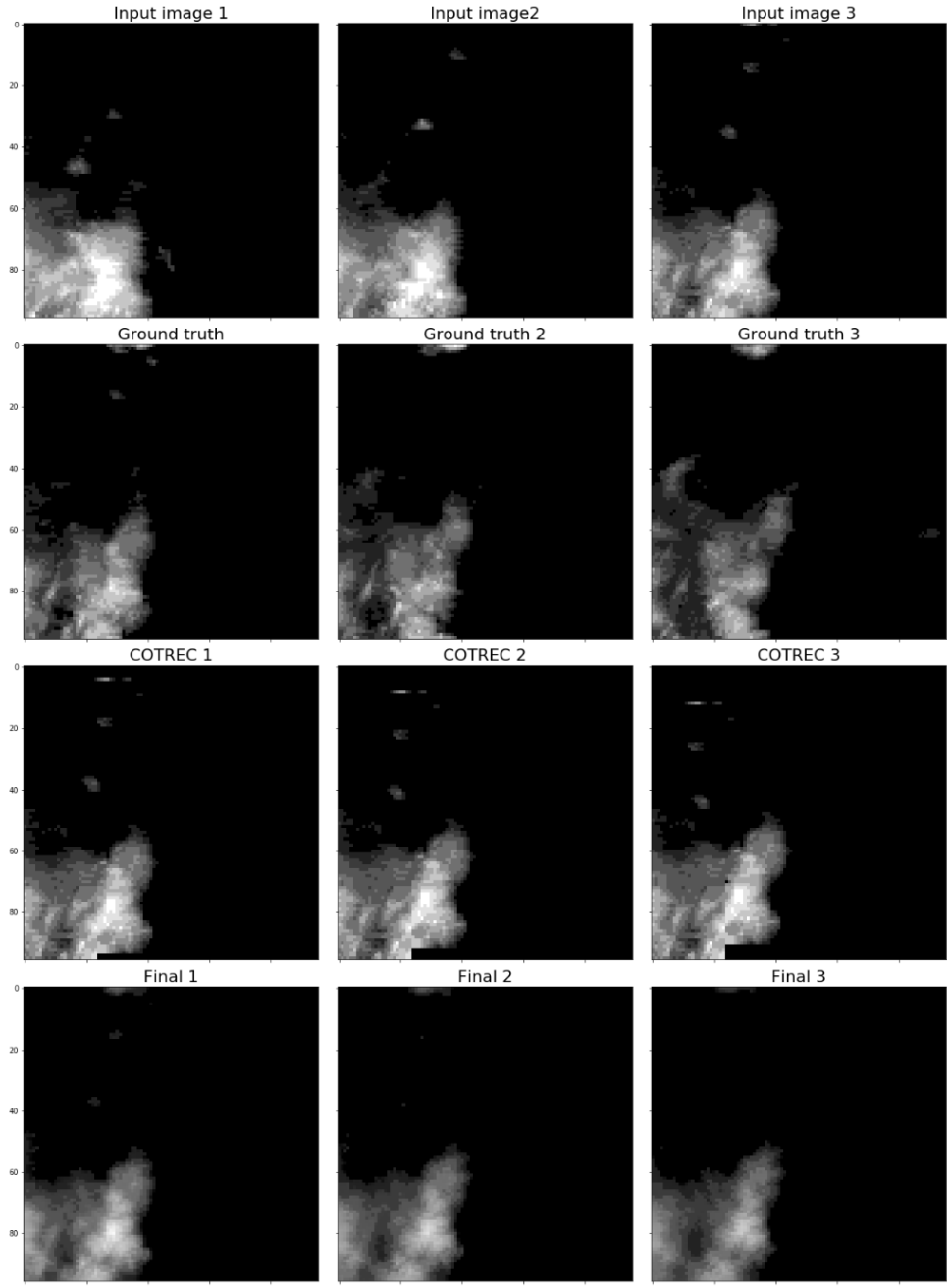


Figure C.3: Note in this example disappearance of the small, low intensity radar echoes in the left top part of the frame, in the output sequence extrapolated with my network.