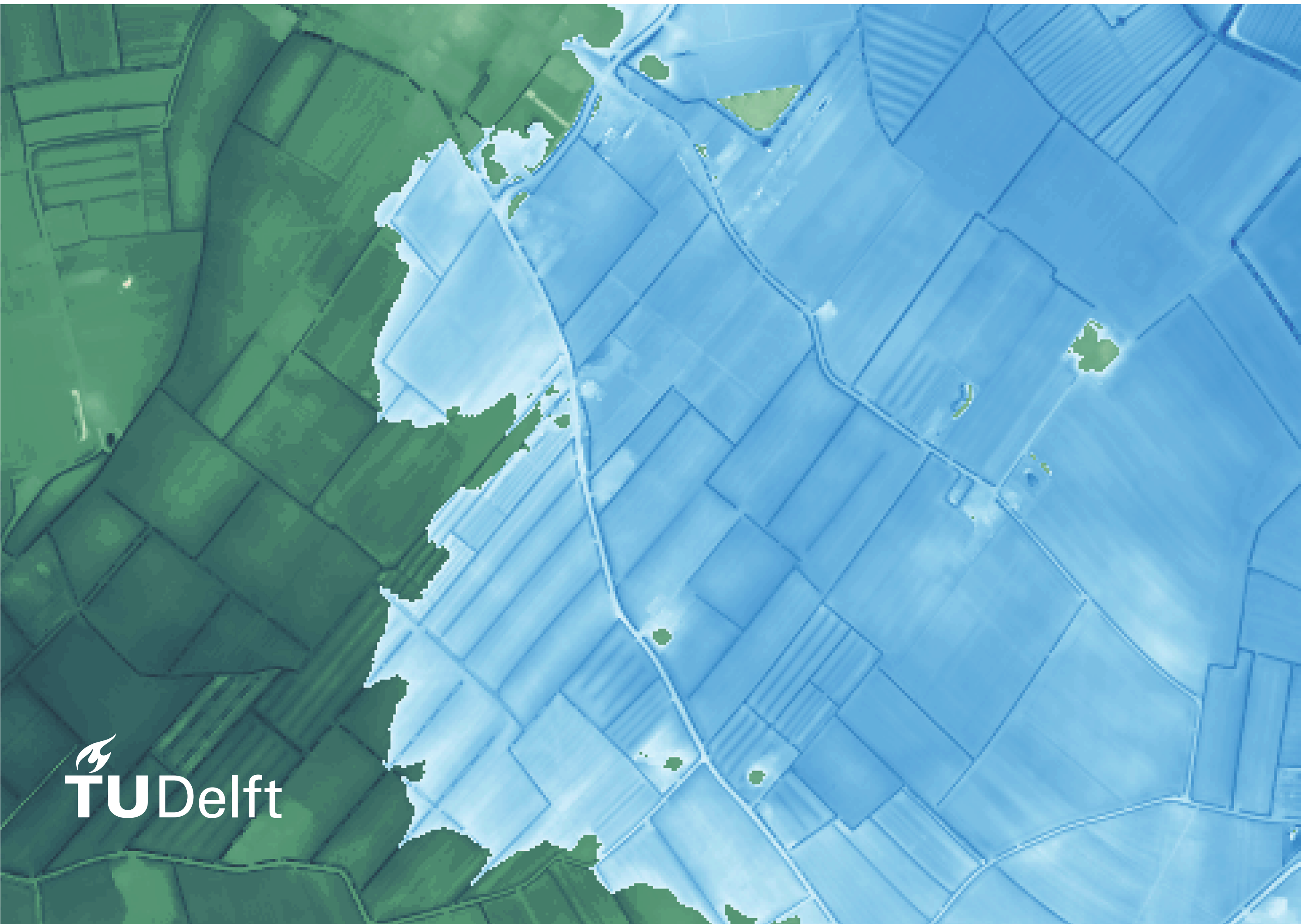
Deep learning-based surrogate modelling for 2D flood simulation

T. Stolp

Hydraulic Engineering



Deep learning-based surrogate modelling for 2D flood simulation

Thomas Stolp

to obtain the degree of Master of Science
at the Delft University of Technology,

Student number:	4377907	
Project duration:	September, 2020 – Augustus, 2021	
Thesis committee:	Dr. J.P. (Juan Pablo) Aguilar-López	TU Delft (chair)
	Prof. dr. ir. M. (Matthijs) Kok,	TU Delft
	Dr. ir. M. A. (Andres) Diaz Loaiza,	TU Delft
	Dr. ir. J. (Jesse) Krijthe,	TU Delft
	Ir. G. (Geerten) Horn	HKV

Preface

This MSc Thesis is the final part of the master Hydraulic Engineering at the faculty of Civil Engineering and Geosciences and also an end of my time as a student TU Delft. This research has been conducted at HKV Lijn in Water as a graduate intern and has been a valuable experience. I would like to express my gratitude to everyone that supported me during the past months.

Firstly, I would like to thank my supervisors for their support and guidance. Juan Pablo, my daily supervisor and chair, thank you for giving me guidance during this period, for the inspiring discussions and for keeping me on the right track. Thanks for making me interested in deep learning applications and for providing the motivational words that I sometimes needed to keep going. I would like to thank Geerten Horn, my supervisor from HKV, for all the support during my time as graduate intern. Our weekly meetings were really helpful, I learned a lot about flood simulations and how they are used in practice. I would like to thank Matthijs Kok for his critical notes and asking the important questions. I want to thank Jesse Krijthe for the discussions and advice in training deep learning architectures and Andres Diaz for his feedback during the meetings the time spend into my thesis.

I want to thank the people at HKV for their help, especially Mattijn Hoek for his help in installing software libraries that allowed me to use the GPU cluster.

I would like to thank my parents and my sister for their help during my time as a student in Delft. Finally, I want to thank all my friends and my roommates Bram, Maarten and Tommy for the mental support and all the great memories.

Thomas Stolp
Rotterdam, August 2021

Abstract

Flood simulations can give insight into the consequences of flood scenario's and can help to create hazard- and risk maps to support decision-making in flood risk management and in crisis management. 2D hydrodynamic simulations give accurate descriptions of the propagation of a flood and rely on advanced numerical methods to solve a set of physics-based mathematical equations. A drawback of these models is that they can be computationally expensive with run times in the order of hours or days depending on the time and spatial resolutions.

In this study we explore the use of deep learning techniques in a surrogate model for 2D flood simulation. We propose and test a deep learning-based surrogate modelling framework that can be used to train a deep learning-based surrogate model. Once trained, the surrogate model can be used as a substitute for the hydrodynamic model with the advantage of being much more efficient in terms of run time and can be of great value in for example crisis situations.

For training, a data set of expensive 2D hydrodynamic simulations was created using the SOBEK software program. Such simulations require a lot of input data, such as input parameter maps specifying the terrain over the computational grid and boundary conditions. To make training data-efficient, a sampling strategy was used for the input of the flood simulations.

Three deep learning architectures were trained and tested. The first two architectures are feed-forward networks and the third architecture is of recurrent network type. These networks contain convolutional neural network (CNN) architectures with an encoder-decoder structure to make patch-level predictions of the flood characteristics in time. These patches contain a small section of the flood prone area and an encoder network is used to extracts coarse feature maps from this data that is then refined by a decoder network to create a prediction of the flood propagation. Using patches has the advantage of making a surrogate model able to create flood simulations over prone areas without restrictions on size or shape by tiling the output patches with flow predictions. Also it allows the surrogate to focus only on regions where the flood has reached and not on the regions where no water has arrived.

It was found that with the recurrent architecture, the surrogate model was most capable of emulating the ground truth flood simulations in the test simulation. This trained network architecture was used in a case study where the surrogate was applied to create flood simulations in a small dike ring in the Netherlands. This shows that the surrogate modelling framework can be used to train a deep learning-based surrogate and, once trained, can be used to create flood simulations similar to hydrodynamic simulations. However, two main challenges were identified in using such data-driven deep learning-based surrogates. Firstly, keeping the predictions of the flood characteristics accurate enough to avoid large error propagation. Secondly, accurately generating large amounts of data from relatively little information present in the boundary condition and terrain.

Executive summary

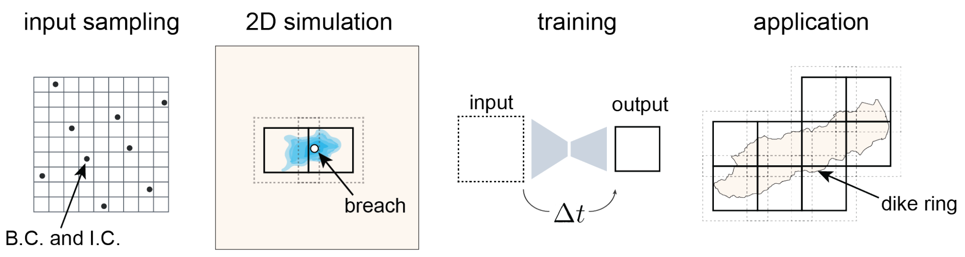
Inundation modelling is an important tool used in flood risk management. Flood simulations can give insight into the consequences of flood scenario's and can help to create hazard- and risk maps to support decision-making in flood risk management and in crisis management. Two-dimensional (2D) hydrodynamic simulations give accurate descriptions of the propagation of a flood. These computer simulations rely on advanced numerical methods to solve a set of physics based mathematical equations called the shallow-water equations. A major drawback of these models is that they can be computationally expensive with run times in the order of hours or days depending on the time and spatial resolutions. Climate change, land subsidence and economic growth bring challenges for flood risk management, which may lead to a greater need for fast inundation modelling tools. This motivates research on more computationally efficient modelling approaches and techniques.

In this study we explore the use of deep learning techniques in a surrogate model for 2D flood simulation. We propose and test a deep learning-based surrogate modelling framework that can be used to train a deep learning-based surrogate model. Once trained, the surrogate model can be used as a substitute for the hydrodynamic model with the advantage of being much more efficient in terms of run time. Being able to create fast surrogate flood simulations can be of great value in crisis situations and in cases where many simulations are required, for example in assessing the uncertainty in the consequences of flood scenario's. The surrogate modelling framework has the following steps:

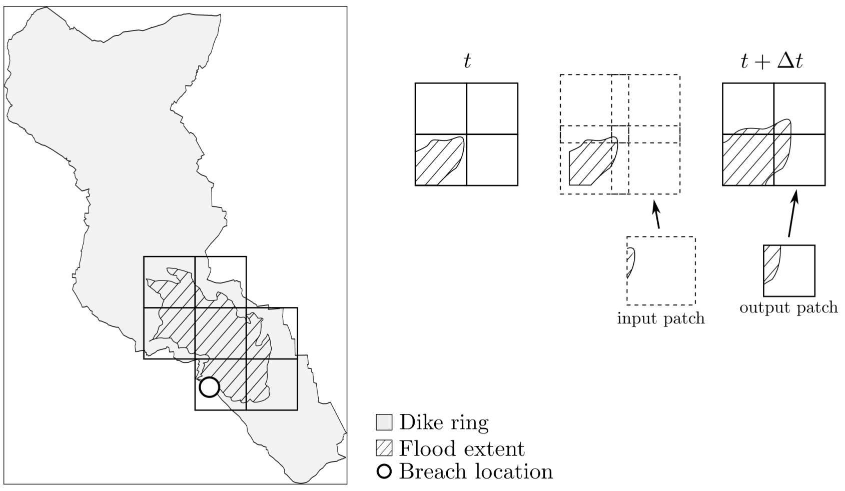
(1) Use input sampling to specify the inputs of a set of fictitious 2D flood scenario's and use these to create a number expensive flood simulations.

(2) Process the flood simulation output to create training examples consisting of inputs and outputs for a deep learning architecture and train this architecture using these examples.

(3) Apply the surrogate model as a tool for rapid inundation modelling and as a substitution for expensive 2D hydrodynamic simulations.



The surrogate model used in this study contains convolutional neural network (CNN) architectures with an encoder-decoder structure to make patch-level predictions of the flood characteristics in time. The surrogate model takes as input a small section of the flood prone area called an input patch. An encoder network extracts coarse feature maps from this data and a decoder network is used to refine these images and create a prediction of an output patch. The input patches contain the flood characteristics, water depths and flow velocities, as well as the elevation and roughness. The output patches contain only the flood characteristics at a next time step. A time shift is between present between the input and output of the network to predict flood propagation in time. Input patches are chosen to be larger than output patches to create overlap regions with neighbouring patches. This gives the surrogate spatio-temporal context and allows flood propagation between patches. Using patches has the advantage of making a surrogate model able to create flood simulations over prone areas without restrictions on size or shape by tiling the output patches with flow predictions. Also it allows the surrogate to focus only on regions where the flood has reached and not on the regions where no water has arrived.



Three deep learning architectures were trained and tested. The first two architectures are feed-forward networks and the third architecture is of recurrent network type. For training, a data set of expensive 2D hydrodynamic simulations was created using the SOBEK software program. Such simulations require a lot of input data, such as input parameter maps specifying the terrain over the computational grid and boundary conditions. In surrogate modelling, these simulation inputs are free design choices. To make training data-efficient, a sampling strategy was used for the input of the flood simulations. A square grid were chosen for the simulations with fixed size. For the terrain, square tiles with matching size were extracted from the Digital Terrain Models (DTM) of a number of dike rings in the Netherlands. Bottom friction of these tiles was modelled with the spatial information of land-use and a conversion table. Simple triangular hydrographs were created for the boundary condition of the simulations which are specified at breach locations approximately in the center of the tiles. The output of these expensive flood simulations consists of large three-dimensional arrays containing water depth and flow velocity information at each time step of the flood duration. A patch-wise training strategy was used and different training runs were performed.

It was found that with the recurrent architecture, the surrogate model was most capable of emulating the ground truth flood simulations in the test simulation. This architecture processes the past flood characteristics and terrain maps with two separate encoder networks and predicts the discrepancies in flood characteristics between the current and next time step. This trained network architecture was used in a case study where the surrogate was applied to create a flood simulation in a small dike ring in the Netherlands. Two main challenges were identified in using such data-driven deep learning-based surrogates. Firstly, keeping the predictions of the flood characteristics accurate enough to avoid large error propagation. Flood are often large scale and have long duration while the time intervals of the simulations are kept small to correctly predict flow between patches. Errors in predictions of the surrogate can quickly add up and cause the simulation to deviate from ground truth hydrodynamic simulations. The trained surrogate model was found to give reasonable predictions up to 10 to 20 time steps. Secondly, large amounts of information have to be generated from the little information present in the boundary condition. This boundary condition is specified as water depth variation in a single grid cell. Accurately generating this a lot of data from this small variation was found to be difficult for the trained surrogate models.

Nomenclature

Acronyms

AE	Autoencoder
CAE	Convolutional Autoencoder
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DTM	Digital Terrain Model
GPU	Graphical Processing Unit
LSTM	Long-Short Term Memory
MSE	Mean Squared error
RMSE	Root Mean Squared error
RNN	Recurrent Neural Network
SWE	Shallow-water equations

Symbols

$\mathbf{I}$	Simulation input	-
$\mathbf{S}$	Sampling plan	-
ϕ	Flood characteristics	-
θ	Angle of rotation	deg
C	Chézy coefficient	$\mathrm{m}^{1/2}/s$
H	Information entropy	-
h	Water depth	m
I_i	Simulation input variable	-
k_n	Nikuradse equivalent roughness	m
R	Hydraulic radius	m
u	Flow velocity in x-direction	m/s
v	Flow velocity in y-direction	m/s

Contents

List of Figures

List of Tables

1 Introduction

Low-lying delta areas such as large parts of the Netherlands are at risk of flooding from rivers as well as from the sea. Flood events can have catastrophic impact on society because they are often accompanied by many fatalities and large economic consequences. In the Netherlands, flood prone areas are protected against flood events by a system of defences such as river dikes and dunes. These system have been divided into several dike rings that are managed using a risk approach. Flood risk is defined as the probability of occurrence of flood events times the consequences of these events. This means that even though flood events may be rare, flood risk can still be significant when the consequences are large such as the case for densely populated area with high economic assets. Quantification of flood risk requires adequate estimating the consequences of flood events for which we often rely on computer simulation.

1.1. Use of flood simulation

Consequences of flood events can be estimated with the use of inundation models and computers that solve the complex numerical systems resulting in flood simulations. The numerical solutions predict the propagation of a flood over a prone area. Factors that influence the extent and its propagation are for example the topography of the terrain and the bottom friction, the presence of dikes and other water retaining elements, the location of the dike breach and the hydraulic load or the volume of water flowing through the breach. With enough input data, flood simulations can be created that estimate characteristics such as the water depth and flow velocities over the flood prone area and in time.

The output of the simulations of certain flood scenario's can be used to create for example hazard maps to show the spatial distribution of the intensity of a flood scenario [25]. These maps show for example the maximum water depths and maximum flow velocities that occur during the flood. This information can serve as input to flood risk assessments. Large flow velocities for example lead to damage to buildings and infrastructure but may also result in more fatalities. Flood simulations may serve as input for damage estimations and cost-benefit analysis to make better informed decisions on for example flood protection measures.

Inundation modelling is also used as an important tool in crisis management. Flood simulations may be used in crisis situations with boundary conditions that apply at that moment in time. These simulations can give an idea of the possible consequences of dike breaches and support decision making concerning application of emergency measures. In addition, the course of the inundation over time can be valuable input for creating more adequate evacuation plans that better apply to the circumstances. With more focused estimates of the flood arrival times and safe areas, it becomes possible to set up more effective mitigation measures.

Another important use of flood simulation and generated flood maps is in spatial design and urban planning. Where conventional flood management strategies have primarily focused on flood prevention, aiming at decreasing the probability of occurrence of flood events, more recent flood risk management strategy aim also at reducing the consequences given a flood. Implementing flood resilience measures such as secondary dikes (dry dikes) and evacuation roads often requires detailed knowledge of flood propagation and flood extent. Flood resilience measures can sometimes be relatively simple to implement but may greatly reduce the overall flood risk. In the Netherlands, the concept of flood resilience has been translated into the multi-layer safety approach [20].

Climate change, expected sea level rise and ongoing land subsidence may result in hydraulic loads with larger magnitude and frequency. Flood prevention may therefore become technically challenging in the future [12]. Spatial planning and flood resilience measures start to become a more embedded policy in flood management strategies in the future. Such approaches however bring the need of assessing the combined effect of these flood mitigation measures leading to a need for fast inundation modelling tools to simulate the effect of these measures on the overall response of the system. As the flood protection system gets more complex, inundation modelling approaches should be computationally efficient. This motivates the study for new techniques that can be used in efficient inundation modelling approaches.

1.2. Problem statement

For accurate and realistic flood simulations, two-dimensional (2D) hydrodynamic models are often used. For creating these simulations we rely on the use of advanced numerical methods to solve a set of physics-based mathematical equations called the shallow water equations. Depending on the simulation size and grid resolution, these simulations can be very computationally expensive. The performance of numerical methods have been improved over the years, for example with innovations in parallel computing and the use of graphics processing units (GPUs) [34], however in practice the run time for simulation can still be in the order of days to weeks depending on the time and spatial resolutions.

The long run time of flood simulations limits the number of flood scenario's that can be taken into account in flood risk assessments. This can make thorough quantification of flood risk challenging, especially for larger flood prone areas. Breach locations often tend to have a large impact of the extend and propagation of a flood and they are often defined based on either historical data or on hypothetical assumptions [37]. The uncertainty of these conditions is often not assessed because of a limited computational budget. The large computational cost of flood simulations also limits our ability to address uncertainties in for example the stability of water retaining elements, breach locations and topography of the flood prone area. Often flood defences are constructed with natural material such as clay or sand which result in large uncertainties about the strength and stability of these structures. Because hydrodynamic simulations are computational intensive, quantifying these uncertainties in for example a Monte Carlo framework is currently unfeasible. In crisis situations, where time for decision makers is limited, rapid inundation modelling tools are of vital importance. The calculation time of accurate hydrodynamic flood simulations is currently too long to effectively support decision making in these situations [22]. Even if computations are finished before the peak water level arrive there may still not be enough time to analyse the results and to act adequately upon.

To cope with expensive simulations, surrogate modelling or meta-modelling is sometimes utilized in which a surrogate model is trained emulate the behaviour of the original simulations. Once trained, a surrogate model can be used to create surrogate simulations while taking only fraction of the calculation time required to make the original expensive simulation. Surrogate modelling has been applied to problems in water resources extensively [28]. Large scale flood simulations require large amounts of input data which is a challenge for using surrogate modelling as the dimensionality of the input space of such simulations can be very high. Also, hydrodynamic models are non-linear and depend on simulation input in a complex way.

Recently, deep learning techniques have been applied to problems and applications involving high dimensional array data or image data. The application of trained deep learning models has been successful in computer vision, which involves finding non-linear relations between in- and output. Such deep learning architectures may also be applied inside a surrogate model for emulating 2D flood simulations. A few deep learning-based surrogate models have been proposed in literature to predict fluvial and pluvial flood propagation [6] [17] [42] [14] [13] [45]. However, the use of a deep learning-based surrogate as a full substitute for 2D hydrodynamic simulation resulting from a dike breach has not been addressed. Most studies focus on emulating the propagation of water depths and do not include the flow velocities that occur during a flood. Also, the choice for appropriate training data is still an open research question.

1.3. Research objective

In this study we explore the use of deep learning techniques in a surrogate model for 2D hydrodynamic flood simulation. The goal of this research is to create a surrogate modelling framework that can be used to train a deep learning-based surrogate model to emulate expensive flood simulations. For constructing this framework we have to give answer to the following research questions:

Q1 Which state-of-the-art deep learning techniques and what architectures can be used in a surrogate model for 2D hydrodynamic flood simulations?

Q2 How can a deep learning-based surrogate model be used to create surrogate flood simulations in flood prone areas such as dike rings.

Q3 How can such deep learning-based surrogate model be trained effectively on flood simulation data and how should this data be processed?

Q4 How do are trained deep learning-based surrogate models perform in application?

We focus on surrogate modelling of large-scale flood simulations resulting from a levee breach. We aim to identify the deep learning techniques that are most suitable for predicting flood propagation and that can produce output that is similar to the output of 2D hydrodynamic simulations. In other words, predict water depths and flow velocities over the flood prone are in a grid like structure. Also, we investigate what flood simulation data can be used for effective training of deep learning architectures and how this data should be processed. We will create 2D hydrodynamic simulations with the SOBEK software program and use this data to create training examples for deep learning models. Different architectures will be trained and their performance on a test simulation will be compared.

1.4. Thesis Outline

In this section we give an outline of the content of the remaining chapters this report. In Chapter 2, we begin with an introduction to inundation modelling and 2D hydrodynamic models. We discuss the SOBEK software program that can be used to create 2D flood simulations and which is used in this study for generating training data. In addition, we explain what initial- and boundary conditions can be specified for these flood simulations.

In Chapter 3, we give an introduction to deep learning and present state-of-the-art techniques and architectures that are suited for use in the surrogate model. Convolutional neural networks (CNNs) are discussed which have been used for image-to-image regression problems and recurrent neural network are discussed which are suitable for problems involving an input sequence.

In Chapter 4 the surrogate model framework is proposed that can be used to train deep learning-based surrogate models to emulate flood simulations. We explain how the surrogate model can be used to create surrogate flood simulations in flood prone areas, we introduce a sampling strategy for sampling inputs of flood simulations used for training data and we discuss how this simulation output can be processed to create training examples suitable for deep learning architectures. Finally, two feed-forward architectures and one recurrent architecture are discussed that will be trained and tested with the use of a test simulation.

In Chapter 5 the training process of the proposed deep learning architectures is discussed. We present two training scenario's and we explain the choice for hyperparameters of the networks. Also, a test simulation is presented that will be used to test the performance of the surrogate model with the various trained architectures.

In Chapter 6, we discuss the performance of the feed-forward network architectures on the test simulation on the test simulation. We compare the results of various training runs and evaluate filters for error reduction.

In Chapter 7, we discuss the performance of the surrogate model that uses a trained recurrent architecture. In addition, we present a small case study in which the surrogate model is used to create simulations for a dike ring in the Netherlands. These surrogate simulations are compared to flood simulations created with SOBEK. We also discuss the calculation time of the surrogate model and compare it to that of the flood simulations created with SOBEK.

Chapter 8 presents the discussion in which the results are interpreted and the relevance of this study for practical use is evaluated. Finally, in Chapter 9 we present the conclusion and the recommendations for future work.

2 Two-dimensional (2D) hydrodynamic flood simulations

In this chapter, we give an introduction to inundation modelling and the types of modelling approaches. We discuss two-dimensional (2D) hydrodynamic models, which is will be used to generate flood simulations in this study. These simulations are what we aim to emulate with the proposed surrogate modelling framework. The inputs of such simulations and other modelling choices are discussed. Finally, we discuss the SOBEK modelling software which is used in this study to create flood simulations.

2.1. Important factors in inundation modelling

Inundation modelling requires a lot of input data to make the flood simulations realistic for the flood prone area being assessed. Figure 2.1 shows a typical situation of a flood resulting from a dike or levee breach. Often the consequences of the event are not the same for each location in the flood prone area. Secondary dikes, sometimes called dry dikes, can be present in the area and potentially stopping the flood or delaying the arrival of the flood. It is important to have accurate schematizations of these obstacles in the used inundation modelling tool to get realistic and credible outcomes. The terrain of the flood prone area should thus ideally be modelled with enough detail. A Digital Terrain Model (DTM) is often used to model the height of the terrain and can be created using LiDAR surveys. Height differences guide the flood over the area, sometimes through ditches or waterways which have relatively low friction and thus can transport water faster. Higher areas will often not be affected severely by a flood, however then can still be reached if for example the flow velocities are large. The boundary condition of a simulation, or the hydraulic load, has arguably the most impact on the flood propagation. Often extreme boundary conditions are determined along a river for different return periods based on statistical extrapolation of a series of historic discharge measurements. In the Netherlands, generated rainfall and discharge extremes are used to estimate extreme boundary conditions for the river Meuse and Rhine (GRADE [41]). The breach location is also an important input of a flood simulation. Often different breach locations are specified in different flood scenarios. Insight into the important factors described above is important for selecting the right set of flood scenario's for which to make expensive flood simulations [8].

2.2. Types of inundation models

Different types of inundation models are available and can be used to create flood simulations and the physical complexity of these models differs. In general, more physical complexity gives more realistic and accurate flood simulations but it comes at the cost of the calculation time. besides complex physical models, also inundation models exist that are not based on physical principles. These models are sometimes referred to as simplified methods [34]. The main purpose of these models is to give a low-detail estimate of some of the state variables, such as the overall extent of a flood event. An example of such method is the bathtub method. These simplified modelling approaches are computational efficient and can thus be used to give insights for large flood domains. Sometimes however, complex topographies are present in the flood prone area which have effect on the flow dynamics making simplified conceptual models less applicable. In these cases inun-

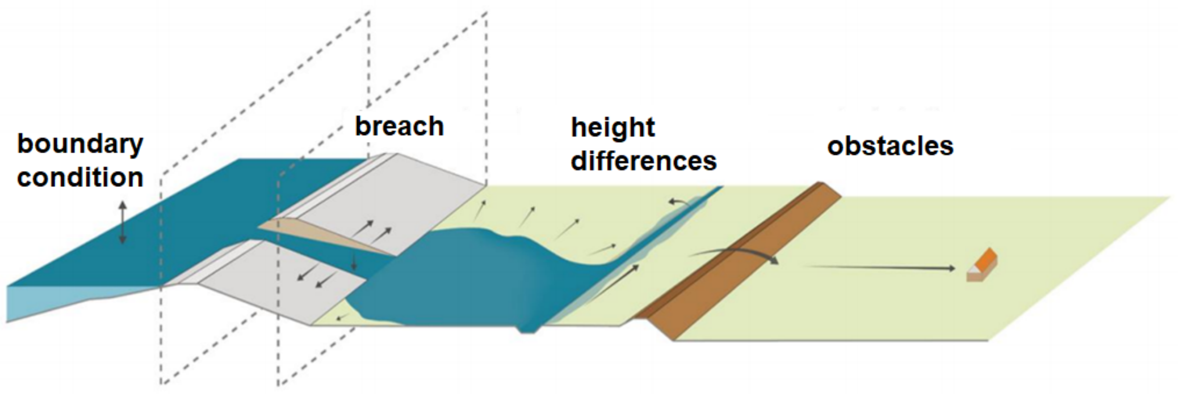


Figure 2.1: **Schematization of a dike breach and the factors that influence the flood propagation** [8]

dation models that are based on physical principles are preferred which are called hydrodynamic models.

Inundation models can be classified based on their spatial dimensionality [38]. In some cases the assumption of a dominant flow direction can be made and a 1D model is accurate enough the describing main features of the flow. This is often the case for flow in waterways and rivers. For the simulation of overland flow, for example in flood risk assessment studies, 2D models are mainly used. The 2D simulations give more reliable results for overland flow than a 1D network, however there is a trade-off between accuracy and run time. A combination of two-dimensional model with one-dimensional elements is also possible, sometimes referred to as 1D2D models [9]. The main advantage of these models is that the 2D grid can be coarse since hydraulic structures and small waterways are represented by the 1D elements, which is computationally efficient an accurate at the same time. Table 2.1 gives an overview of different inundation modelling approaches and how these roughly compare in terms of run time and accuracy [8].

Table 2.1: **Different modelling approaches for flood simulation and their properties**. Indicative overview of how different inundation modelling approaches compare [8]. Here + indicates applicable, +++ indicates very applicable and - indicates less applicable.

Model type	run time	Accuracy
Bathtub	+++	–
1D	++	–
Quasi-2D	++	-
2D	-	++
1D2D	+	++

2.3. 2D hydrodynamic models

Two-dimensional (2D) hydrodynamic models are based on the shallow water equations (SWEs), which are a set of coupled partial differential equations (PDEs) describing shallow-water flow. These equation arise when making assumptions that the pressure distribution is hydrostatic, which means that there are no accelerations in the vertical [38]. It is used to model also dam break induced flow, even though some assumptions are violated, it can still give a reliable average description of most important flow patterns.

$$\frac{\partial \zeta}{\partial t}+\frac{\partial hu}{\partial x}+\frac{\partial hv}{\partial y}=0 \tag{2.1}$$

$$\frac{\partial u}{\partial t}+u\frac{\partial u}{\partial x}+v\frac{\partial u}{\partial y}+g\frac{\partial \zeta}{\partial x}+g\frac{u}{C^2 h}\sqrt{u^2+v^2}+au|u|=0 \tag{2.2}$$

$$\frac{\partial v}{\partial t}+u\frac{\partial v}{\partial x}+v\frac{\partial v}{\partial y}+g\frac{\partial \zeta}{\partial y}+g\frac{v}{C^2 h}\sqrt{u^2+v^2}+av|v|=0 \tag{2.3}$$

The momentum equations include a term for the local acceleration, two convective terms, a term for the horizontal pressure gradients and two friction terms for wall and bottom friction. u denotes the flow velocity in x-direction, v denotes the flow velocity in y-direction and h is the total water depth. The water level above plane of reference is denoted with ζ, C is the Chézy coefficient and α is the wall friction coefficient. Figure 2.2 shows the definition of the water depth and water height with respect to a certain reference level.

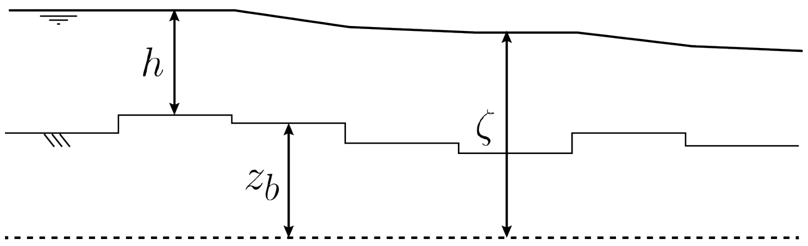


Figure 2.2: **Definition of water height ζ.** The water height equals the height of the water above a certain reference level and can be decomposed into the bed elevation z_b and water depth h.

2.3.1. Computational grid

The 2D shallow water equations are a set of coupled non-linear equations and in order to find solutions of these equations one has to rely on numerical simulation as the system is to complex for analytical solutions. In numerical simulation, the continuous space of the mathematical model is approximated with a discrete space consisting of a finite number of grid points such that solutions can be stored in computer memory [44].

There are many ways in which a computational grid can be constructed. The most straight forward way is perhaps to define a grid of rectangular cells with sizes Δx and Δy, sometimes called a Cartesian grid or uniform regular grid. Sometimes the terrain of a flood prone areas is relatively homogeneous. In for example flat areas a very fine computational grid may not lead to significantly different flow patterns then a courser grid. The use of a computational grid with a fine resolution does however lead to significantly longer calculation times. Innovations in constructing numerical grids allow courser cells in these areas whereas they use grid refinement in areas with a lot of height difference or obstacles, allowing faster computation. These unstructured grids are referred to as flexible meshes.

2.3.2. Numerical methods

With a computational grid, the spatial derivatives in the shallow water equations can be approximated. There are different ways to approximate these derivative terms, such as finite element and finite volume, but the most common method is by approximating the spatial derivatives with finite differences [38]. This approximation of the spatial derivatives with finite differences is called the semi discretization step. The result of the semi discretization step is a (often large) system of ordinary differential equations.

After the discretization in space follows a time integration where the time derivatives are approximated with either explicit or implicit methods. This two-step process is sometimes called the Method Of Lines (MOL) and this method results in a (large) system of algebraic equations, where the number of equations equal the number of grid cells in the computational domain. The algorithmic way of computing the numerical solutions is sometimes called the numerical scheme or numerical recipe. Important properties of schemes are accuracy and stability. The stability of a numerical method often brings a restriction on the choice of time step and grid size. Often the stability condition can be expressed using the Courant number σ.

$$|\sigma| = \frac{|u|\Delta t}{\Delta x} \leq 1 \tag{2.4}$$

2.3.3. Simulation output data

Because the shallow water equations are solved numerically, the solutions are provided on the computational grid that has been provided as input. The numerical solutions have grid-like structure, for Cartesian grids it is very similar to image data. Because floods propagate in time, the simulations have a temporal dimension and often images are produced at regular time intervals. The simulation for a single time step is referred to as a snapshot of the flood and denoted ϕ_t, which is a vector containing the flood characteristics defined on the grid points of the computational domain. For 2D hydrodynamic models, $\phi_t = (h_t, u_t, v_t)$ consist of the water depth h_t and flow velocities u_t and v_t.

.

2.3.4. Terrain models

The reliability of 2D inundation models relies heavily on accurate description of the terrain of the flood prone area. For large floods, the overall flood extent can sometimes be predicted reasonably well with coarse terrain data because the water level tends to be larger than local terrain features. On the contrary, when topographic features, such as secondary dikes and flood retaining structures have large effect on the flood propagation and extend, relatively fine resolution terrain data is required [3]. There are different survey techniques available for topographic data gathering. Airborne light detection and ranging (LiDAR) is a preferred survey method because it allows to separate buildings, structures and vegetation from the land surface and because of the horizontal resolution and vertical accuracy [31]. In some countries, accurate LiDAR elevation data is available. In the Netherlands elevation data from AHN (actual height data bank of the Netherlands) is often used, which provides elevation data with horizontal resolution up to 0.5 m. The raw data from LiDAR surveys can be processed for use in inundation models in various ways. Often the bare-earth elevation is extracted only, neglecting the height of buildings and other structures.

In addition to the DTMs also the surface friction of the terrain can be modelled and given as input for flood simulations. The bottom friction affects the flood propagation and flood arrival time of the inundation throughout the flooded area [40]. A common way of determining the spatial-distributed friction coefficients is by using land-cover or land use information in combination with a conversion table, such as given in [8]. In this way, a friction grid can be constructed having the same resolution as the DTM used in a flood simulation.

2.3.5. Boundary conditions

The hydraulic load or boundary condition of a certain flood scenario is also an important input required to make a flood simulation. In the context of floods resulting from a levee breach, the breach location in the computational domain and a certain discharge or water level function can be specified. In reality the breach location can be anywhere along the of the dike. Often the hydraulic boundary condition is specified by means of a hydrograph which specifies the discharge or water level in the adjacent river as a function of time. To get a realistic boundary condition, the adjacent river can be schematized with a 1D or 2D model such that water water flows from the river through the breach. Water levels in the river will drop and the amount of water that flows into the flood prone area will be reduced.

In the Netherlands, floods are due to breaches in the primary defence of levee systems. These breaches are typically the most uncertain part of the inundation. The location of the breach is the first unknown and is very difficult to predict as it depends among others on the soil properties of a section of dike. There are different approaches that one can take in modelling the breach development, that is, the growth of the width of the breach as well as the depth of it. In the Netherlands, the breach growth rate formula developed by Verheij and der Knaap [35] is often used. This breach development formula describes the width an depth of the breach over time. As a maximum width, 75 meter for clay is taken and 200 meter for sandy dikes. Often it is assumed that the breach reaches a maximum depth in 10 minutes.

One way of defining a boundary condition in SOBEK is with by choosing a 2D boundary node where a time-varying water level $h(t)$ or a discharge $Q(t)$ is specified. A 2D grid cell that contains a boundary can only discharge into one of the four neighbouring cells. This is different from the grid cells that do not contain a boundary node, which can discharge into any of the neighbouring cells. By default it discharges into the cell directly right to it [15].

2.3.6. Inundation modelling software

There are numerous inundation modelling programs available for making hydrodynamic flood simulations. Examples are SOBEK, HEC-RAS and D-Hydro. In the Netherlands, most existing flood simulations are either 2D or 1D/2D simulations made with SOBEK1D2D. So called 0-order models or bathtub models are in some cases used to make flood simulations for smaller dike rings and floods from regional waterways. In the river floodplains, the WAQUA models are often used. DFlow-Flexible Mesh (DFlow-FM) is a software program that uses the innovations in unstructured grids or flexible meshes and was developed by Deltares and the Delft University of Technology and uses the numerical concepts of SOBEK1D2D [18].

2.3.7. SOBEK

In this study, the SOBEK software program is used because it can be used to write simulation output text files of the flood characteristics every time step (1 minute), which can be used as training data for deep learning architectures. In addition, it uses a rectangular grid which is easy to construct and results in output that has uniform structure which is easy to interpret. The SOBEK model was developed by Delft Hydraulics, now Deltares, and has played an important role in quantification of flood risk in many projects in the Netherlands. One example of the application of SOBEK is in The Flood Risk in the Netherlands project (Veiligheid Nederland in Kaart, in Dutch), which is a large-scale assessment project of flood risk in the Netherlands and was started in 2007 [16]. SOBEK is a 1D/2D modelling framework which gives fully implicit finite difference scheme for the 2D shallow water equations. The convective momentum terms in the equations are in a special way such that mixed sub- and super critical flows can be computed [2]. This discretization makes the model robust and accurate and also damp the oscillation in velocity directions at irregular model boundaries. At each time step, checks are made to ensure only physically realistic results are output of the model. The time step is reduced such that no negative water depths can be present in the approximations. For flooding and drying, a similar approach is used where each time step only a maximum of one neighbouring cell is allowed change from a dry to a wet state or the other way around.

3 Deep-learning architectures for the surrogate model

In this chapter, we give an introduction to deep learning and introduce the techniques and architectures that can be used for a surrogate model of flood simulation. For a more detailed explanation of this topic of deep learning, the reader is referred to [11]. Firstly, we will discuss convolutional neural networks (CNNs) and their use in encoder-decoder networks for image-to-image translation problems. Secondly, we discuss recurrent neural networks (RNNs) and how they can be combined with CNNs to handle inputs containing a sequence images.

3.1. Introduction to deep learning

Deep learning generally refers to machine learning models with a layered architectures that are ’deep’, or in other words, that contain many layers stacked on top of each other. Most layers of these network contain trainable weights that are adjusted in the training process to make the model fit output data well. Activation functions are applied to the output of the layers, which are often non-linear functions such that deep learning models are flexible and able to learn complex mappings between the input and output. A key advantage of deep learning over classical machine learning methods is that feature selection from data is incorporated in the learning process [21]. Each layer may extract different information or features from the input, and with many layers the network may be able to detect higher level representations. A first layer of a trained network may for example detect edges in an input image, whereas layers that are located deeper in the network may detect larger motifs.

The network complexity separates a deep learning model from a larger family of machine learning models that have been around now for a few decades. In both approaches, the networks are trained to find relations between inputs and outputs rather than using rules that were explicitly formulated. Deep learning techniques have recently gained popularity in many applications and are also starting to be widely used in the water industry [33]. The popularity of these techniques is primarily due to the availability of enormous amount of data. Furthermore, development in computer resources contributes to this trend. Deep-learning software libraries such as Tensorflow and Pytorch are freely available and open-source. These libraries take advantage of the computing power of modern graphical processing units (GPUs).

Networks in deep learning are often referred to as neural networks as they are loosely inspired by the connections between neurons in the human brain. The activation functions of the network layers have conceptual similarities to a process in neuroscience called action potential and give deep learning models the ability to learn non-linear mappings. The nodes in the networks are sometimes called neurons. The simplest type of neural network is the fully connected network in which all nodes in a layer are connected to every node in an adjacent layer. Other versions are recurrent neural networks (RNNs) that are frequently used in natural language processing tasks and convolutional neural networks (CNNs) that have shown to be effective in for example image classification tasks. These types of networks will be discussed in more detail in Section 3.2.

Often, deep learning networks are trained with input- and output examples, which is sometimes called a supervised learning problem. The input examples are often denoted X and the ground truth output Y. In training a neural networks, first the weights are initialized and the network is passed examples in a training set resulting in predictions $\hat{Y}$. A loss function L is used to compute the error between $\hat{Y}$ and Y given X. Many loss functions exist such as the mean squared error (MSE) and mean absolute error (MAE). In training, the gradients of the weights of the network with respect to the loss function can be computed using the chain-rule for differentiation, this process is called a backpropagation step. Adjusting these weights to minimize the loss function is called gradient descent. The learning rate is a value that controls how much the weights are adjusted each training step and is an important hyperparameter of the model.

Apart from the training data, a validation data set is used to evaluate the performance of the model during training process. A typical problem with deep learning models is that they have the tendency to start overfitting, meaning that they start performing very well on the training data but loose the ability generalize and start performing worse on unseen data (see figure 3.1). Evaluating the model on validation data helps to identify overfitting during training. After the model is trained, the model performance can be evaluated by applying the model to unseen data in the test set.

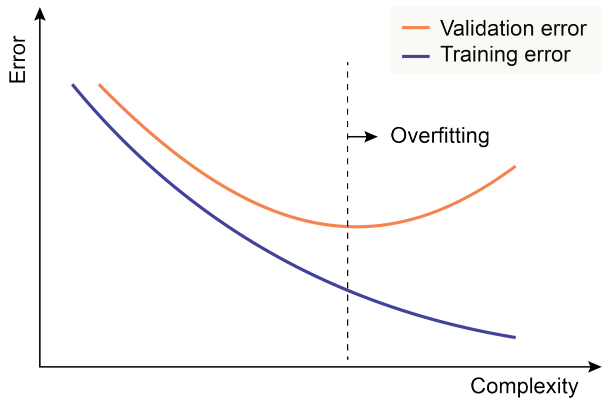


Figure 3.1: **Concept of overfitting**. During training of a deep neural network, the error on training data will decrease. At some point in time the network will start overfitting on the training data examples, which can be spotted by keeping an eye on the validation error. The error on validation data will increase as the network starts to focus too much on the details of the training examples.

3.2. Convolutional Neural Networks

Image data is typically high dimensional data with grid-like structure. A deep learning architecture that was specifically designed to handle this data format is called a convolutional neural networks (CNN) or ConvNet [21]. CNNs contain among others convolutional layers that consist of a kernel of trainable weights that performs a convolution operation on the input. These layers learn a feature representation of the input. A kernel convolves over the input array (or image) and by doing so the kernel weights are multiplied with the underlying input elements. This operation results in an output array to which then an element-wise (non-linear) activation function is applied.

Apart from convolutional layer, CNNs often also contain other types of layer such as layers that reduce the dimension of the input which are called pooling layers and layers that increase the dimension of the input, upsampling layers. Pooling layers are layers that reduce the dimension of the input. They have kernels and make a convolution over the input, similar to convolutional layers, however do not contain weights. Instead, they output values using some criteria such as taking the average of the inputs under the kernel or take the maximum value. Upsampling layers increase the dimension of the input and thus are a type of layer that can be used to generate data. This generation of data is done by using an interpolation method such as nearest neighbor, bicubic or bilinear interpolation.

One may ask the question what makes these type of network so successful in deep learning tasks. There are three main characteristics of CNNs that set them apart from other network types. Firstly, convolution layers have sparse connectivity (sometimes called sparse weights or sparse interactions) [11]. In dense layers or fully connected layers output nodes are connected to every input node. Using these types of layers for high dimensional inputs, such as image data which can have thousands or millions of pixels, the number of weights in the network will explode. The kernels in convolutional layers are smaller than the input size and thus less weights are used. Due to the convolution operation, the small set of trainable weights in a kernel are applied to all pixels of the input image as apposed to having a unique weight for every input node. These layers are therefore said to be weight sharing. The number of weights that have to be stored is thus much lower which makes convolutions layers much more efficient than dense layers in terms of the required memory. This is inline with the idea that learned kernels that are effective in detecting, say certain edges, in one part of the image are also likely to be useful in other parts of the image. It means that advantage of convolution layers over regular dense layers is that the number of connections is fixed and does not depend on the size of the input array. Finally, the way in which parameter are shared in a kernel gives convolutional neural networks the property of being equivariant to translation. Features that are identified in feature maps can move with the input. This is for example very important in object detection, where an object should be given the same label irrespective of its location within the image.

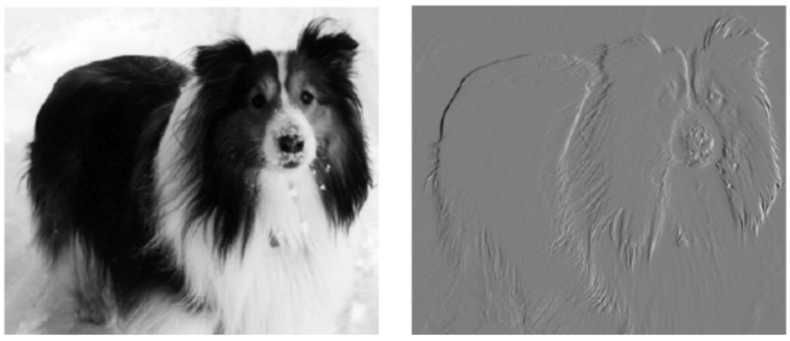


Figure 3.2: **Edge detection in an image.** Detection of vertical edges can be done using a convolution operation with a kernel of two elements. The same transformation can also be described by a matrix multiplication, however, it is much less efficient in representing and computationally. It is a transformation that involves applying the same transformation of a small region over the entire input, which is done efficiently by convolution. [11]

3.3. Recurrent neural networks

It was discussed that convolutional neural networks are typically used for data that are in a grid structure, such as image data. A type of networks that is used and specialized for sequential data are referred to as recurrent neural networks (RNNs). This type of network processes the inputs of the sequence once at a time and they keep track of a state variable that is sometimes called the memory of the network as it can contain information from inputs earlier in the sequence [11, 21]. Training these type of networks is sometimes difficult, however improvements such as Long-Short Term Memory (LSTM) networks partially solved these training issues.

3.4. Encoder-decoder network

Apart from supervised learning, where labels are given to the model explicitly, another type of learning for deep learning models is called unsupervised learning. With unsupervised learning, the labels are not explicitly given to the model, in contrast to supervised learning, but the input data itself is used for extracting features. The model is forced to learn a representation of the data based on some global criteria and this representation can than be used for all sorts of tasks, such as dimensionality reduction transformations [11].

Dimensionality reduction techniques have been used in extraction of flow features from large fluid mechanics simulations [4]. In this field, the simulation outputs are often two- or three-dimensional and can be very large. In dimensionality reduction, one tries to find lower dimensional representations of this data often using an information bottleneck. A well known technique is called linear principal component analysis (PCA). The information bottleneck is usually a vector with entities called latent variables. Autoencoder can be seen as a deep learning dimensionality reduction technique, or latent variable model, and they can introduce non-linearity making them more flexible [1]. An autoencoder network is a deep neural model consisting of two network, an encoder that tries to encode the original information into a latent space, and a decoder network that aims to reconstruct the original information. The encoder and decoder are both neural networks consisting of multiple layers, such as convolutional layers, with trainable weight and they are trained using only the training data itself and without any additional labels. The loss function which is used as a global criterion for this learning process is called the reconstruction loss. If autoencoders are applied to images, a pixel wise difference between the input and reconstruction can be taken as a reconstruction loss function.

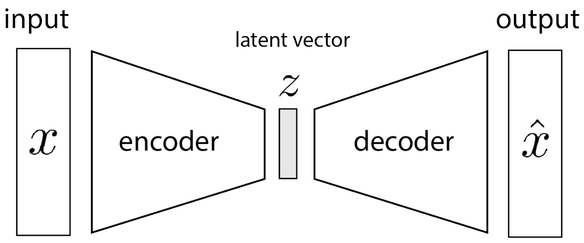


Figure 3.3: **Autoencoder network for dimensionality reduction**. In a self-supervised learning setting, the encoder-decoder structure aims to encode the input to a lower dimensionality latent space variable after which the information is decoded to reconstruct the original input.

Figure 3.3 presents a illustration of the concept of an autoencoder network. Here x denotes the input of the network and $\hat{x}$ the output or reconstruction. The latent space variable is of lower dimension than the original input, creating a dimensionality-reduction or compression of the input. Here the latent variable is a vector. In the encoder and decoder networks, fully connected layers can be used. When autoencoders are used for reconstruction image data, usually they consist of convolutional layers. The encoder reduces the dimension of the input and thus pooling layers are used in this network. The decoder generates data and thus upsampling layers are used there. These encoder-decoder type networks with convolutional layers are often called convolutional autoencoders (CAEs), originally proposed by [24].

3.5. image-to-image regression

In some problems in deep learning tasks, the input of and output of a learning problem are both images and the goal of training a network is to learn a specific translation from the input- to the output image. This problem is different from image classification because now pixel-wise predictions are learned where a quantity is predicted is made for every pixel in an image instead of prediction a label for an image. For these image-to-image regression problems, a coarse-refine process can be used where high-level coarse features are extracted from the input images with an encoder network and a decoder network is used to refine the coarse features to produce an output image [43]. The encoder-decoder structure is similar to autoencoder networks, however the input and output images of image-to-image regression problems may appear very different while with autoencoder networks the goal is to reconstruct input images.

3.5.1. Convolutional Encoder-decoder network

For image-to-image regression problems, CNN-based networks can be used. The encoder network contains convolutional layers and pooling layer to extract features with reduced dimensionality. The decoder network contains convolutional layers and upsampling layer to refine the course features and produce an output image. In solutions of partial differential equations (PDEs) a time dimension is often present. The input and output images of the image-to-image regression problem have a time shift of Δt. Figure 3.4 illustrates an such a problem formulation where a convolutional encoder-decoder network takes as input a snapshot of some physical quantity ϕ_t and as output the next snapshot in the sequence $\phi_{t+\Delta t}$.

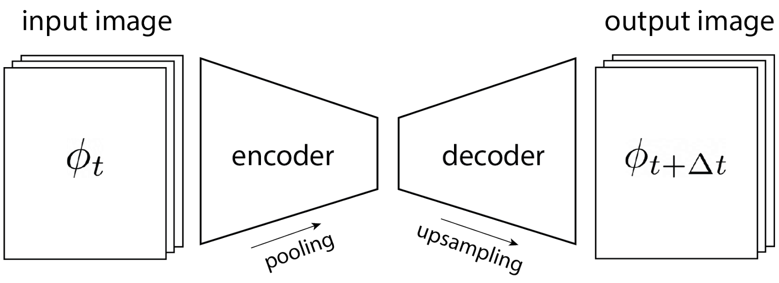


Figure 3.4: **Time-lagged autoencoder structure**. This architecture consists of an encoder network that forces the input into a lower dimensional latent space vector or matrix, and a decoder that constructs a target from this latent space. In this case the target is a time-lagged version of the input. [39]

3.5.2. U-net

The course-refine process of the encoder-decoder network causes some information loss and can make the output images appear blurry. Sometimes, so called skip connections are added to this architecture to reduce the loss of information through the bottleneck. These skip connections are paths where feature maps are copied from some layer in the encoder to a location in the decoder. Network architectures with these skip connections are referred to as U-net architectures and were first proposed by [29], which used such network for image segmentation task in a biomedical context. Adding these skip connections can be effective when the input and target of the network contain similar information at different spatial scales [23]. When input and output of a network may be very similar, for example if they are snapshots with a small time shift Δt such as in presented in Figure 3.4. In these situation, skip connections may be very effective to reduce the information loss caused by pooling and upsampling.

3.5.3. Convolutional LSTM

In addition to feed-forward networks architectures also recurrent networks can be used for image-to-image regression problems. For example, a sequence of input images or snapshots can be provided when the data has a temporal dimension. A convolutional LSTM layer is a layer type that combines the convolutional layer with a LSTM layer. The input-to-state and the state-to-state transitions have a convolutional structure instead of a fully connected (dense) connection [32]. Figure 3.5 presents the structure of a convolutional LSTM architecture.

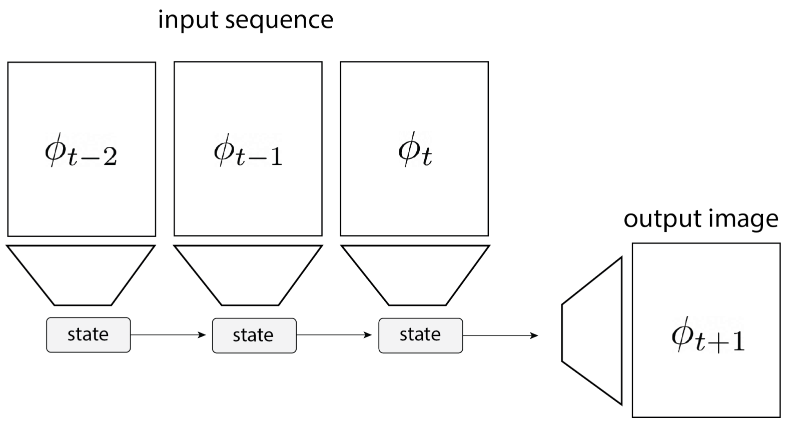


Figure 3.5: **Recurrent encoder-decoder structure**. This type of architecture uses the encoder to extract the most important features from a sequences of inputs and one or more recurrent layers to account for the order of the data within the sequence. The recurrent layer keeps track of a cell state as a type of memory.

4 Surrogate modelling framework

In this Chapter, a surrogate modelling framework is proposed which can be used to train deep neural networks to emulate 2D flood simulations. We give an introduction to surrogate modelling and present an overview of the proposed framework. After that, we discuss how to generate flood simulations by sampling input variables. Next, we present how we use a deep learning-based surrogate model to emulate flood simulation. Finally, we explain the approach for creating training examples from simulation output and we discuss how these examples can be used to train deep learning architectures.

4.1. Introduction to surrogate modelling

Computer simulations of real (physical) systems are used in many fields of science and engineering, for example in weather forecasting or in the design of aircraft. Creating these computer simulations is often computationally intensive, therefore a commonly used approach is that of surrogate modelling where part of the computational budget is invested in creating training simulations that are then used to train a surrogate model [10]. The surrogate model aims to emulate the original expensive simulations and, once trained, it can be used as a substitute for the full order model. The advantage of using the surrogate model is that it is much more efficient in terms of run time. In engineering design the use of such surrogate models can be very valuable, for example in exploring different design options. With the same computational budget of one full order model simulation, multiple surrogate simulations can be created instead. These surrogate simulations may give insight of the effect of multiple designs elements. There are challenges in applying surrogate modelling, for example it is often not so straightforward to choose which simulations are best to use as training data.

A data-driven surrogate modelling approach involves creating expensive simulations and using this data to fit a certain approximation function. This method is referred to as response surface modelling [28]. Another approach is to use simpler physics-based versions of the original simulation model. In the context of inundation modelling, one can think of using simplified models such as the bathtub model or 1D hydrodynamic models as a surrogate for expensive 2D hydrodynamic models. In this study, we use a response surface modelling approach for emulating 2D flood simulations. The main steps of this type of approach are [10]:

(1) Create a sampling plan for input variables, or design variables, of the training simulations. In surrogate modelling one is free to choose and define these inputs. The choice of which simulations to use for training the surrogate model is important because it will influence the performance in the application stage. It will be more challenging for the surrogate to emulate simulations with inputs that differ from inputs of the training simulations. The sample of input variables should ideally be space-filling such that it captures a wide range of responses of the underlying simulation model.

(2) Choose a function approximation model. The model should be flexible enough to approximate the possibly complex response surface of the underlying expensive simulation model.

(3) Train the surrogate model on the set of training simulations. This step may involve adjusting certain model hyperparameters to get a better fit and testing the evaluation of the surrogate model on validation observations to avoid overfitting.

4.2. Framework overview

In this section, we describe the steps of the proposed surrogate modelling framework for emulating 2D flood simulation with deep learning techniques. The frameworks gives answer to the question of what flood simulations to use for training data and how to process the simulation output to make it suitable for training the deep learning architectures proposed in Chapter 3. The steps of this framework are the following.

(1) Use a sampling strategy to define the inputs of a set of fictitious flood simulations that are used for training a deep learning architecture. These input variables I_i are used to specify the initial- and boundary conditions of the simulations, such as the Digital Terrain Model (DTM) of the flood prone area, the bottom roughness map, the location of the breach and the water level hydrograph at that location. Flood simulations can then be created for these scenario's with hydrodynamic modelling software such as SOBEK.

(2) Process the flood simulation output and extract input- and output patches with time shift Δt. The input patches include the flood characteristics ϕ_t at a time t and local terrain maps while the output patches only contain flood characteristics at the next time step ϕ_{t+1}. In this way, examples are created for a deep neural network to learn the flood propagation between two successive time steps taking into account the local topography and roughness. The time shift is present between in- and output patches such that the network is trained to emulate the spatio-temporal behaviour of the flood.

(3) Apply the surrogate model as a tool for rapid inundation modelling and as a substitution for expensive 2D hydrodynamic simulations. This step first involves constructing a static grid of input- and output patches, defining the flood duration and boundary condition. Given the initial conditions, the trained network may then be used to predict output patches at future time steps. By choosing the dimension of input patches to be larger than the output patches, we give the surrogate model spatio-temporal context needed to accurately predict the inter-patch propagation of a flood.

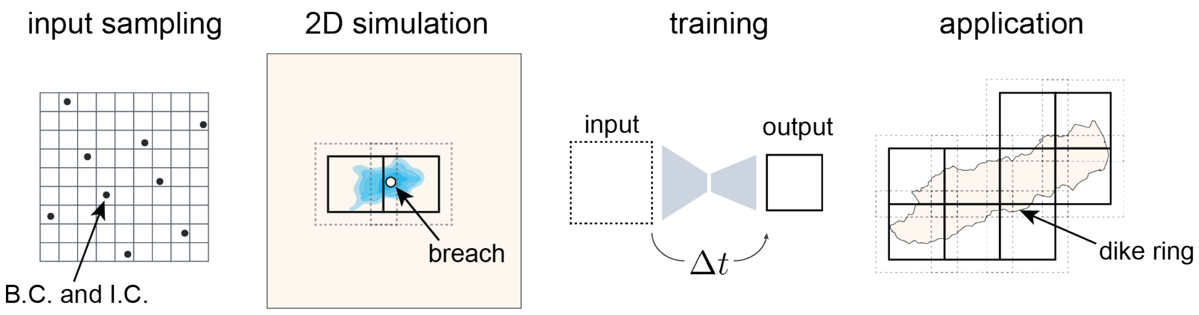


Figure 4.1: **Schematization of the steps of the proposed framework for surrogate modelling of 2D flood simulation**. These steps involve input sampling and creating flood simulations, processing this data to create training examples for a deep learning architecture and applying the surrogate model for rapid flood simulations.

4.3. Generate 2D flood simulation data

In this section, we discuss how to generate the flood simulations that are used as training data for the surrogate model. To create these flood simulations, the initial- and boundary conditions of the simulations can be defined by sampling certain input variables. Here we discuss the choice for the input variables that are used in this study and present a sampling strategy for sampling these simulation inputs.

4.3.1. Flood modelling software

There are many inundation modelling software packages available which may be used to create 2D flood simulations for a training data set. In this study we used SOBEK216 developed by Deltares. One reason why this software program was chosen is that it has been used many projects by the consulting firms around the world and has made an important contribution in many flood risk assessment studies. SOBEK also uses a Cartesian grid, therefore it is easy to process the data. In addition, it provides the option of writing text files with snapshots of the flood characteristics at an output frequency of 1 minute, stored in the ASCII format. These files can be used as training data for deep neural networks. While SOBEK internally can adjust the time step to ensure numerically stable solutions, the simulation output is fixed at 1 minute time intervals.

4.3.2. Simulation inputs

Computational grid

To create a flood simulations, a computational grid has to be defined that covers the flood prone area. In surrogate modelling, we are free to choose which flood prone areas to use for these simulations. In this study, we choose to use a square grid domain with a size of 1000 by 1000 grid cells for all flood simulations. Using the same grid size for each training simulations is convenient for processing the resulting data in a later stage. Because we want to extract (square) patches from the simulation output the computational domain was also chosen to be a square with a size such that multiple patches fit inside the domain with some overlap. For the resolution of the grid, a 5 by 5 meter resolution was chosen. This is a relatively fine grid resolution and therefore certain obstacles such as dikes and waterways are present in the DTM and do not have to be modelled as line elements in SOBEK.

Elevation map

Flood simulations require a model of the terrain of the flood prone area. In this study we use a DTM obtained from Light Detection and Ranging (LiDAR) topographic surveys, which are publicly available for the Netherlands as raster files in the AHN (Actueel Hoogtebestand Nederland in Dutch). This DTM is a two dimensional array containing the elevation above a certain reference level. We used AHN data of a number of major dike rings in the Netherlands and extracted non-overlapping square tiles of size 1000 by 1000 grid cells from these systems. In extracting these tiles, a small overlap with the boundary of the dike rings was allowed. Also a random search was used to find an tiling where much of the area of the dike ring was covered.

As input variable for the elevation map, we use a statistic called the information entropy of the square tiles. Information entropy H is defined as:

$$H = -\sum_i p_i \log p_i \tag{4.1}$$

Here, p_i is the number of times that the elevation of a grid cell falls into the intervals of the ith bin. The entropy of a tile with elevation data is a value that tells something about the level of information or uncertainty of that tile. If a tile is nearly flat, the entropy of this tile would be very low. A tile that contains a lot of variation in height, for example hills, would have a very large entropy value. Entropy is different from the standard deviation of a tile. To see the difference between these two statistics, imagine a tile that is very flat except for one cell that has an elevation much much larger than average. The standard deviation or spread of that tile would be large, however from the definition of entropy it follows that the entropy of the tile is still low. A motivation of using the entropy as a statistic in the random sampling of inputs is that we assume that floods on flat tiles propagate different from floods on a tile with more hilly terrain.

Figure 4.2 shows the DTM of some larger dike rings in the Netherlands which is used in this study (see Table A.2). These levee systems were selected because they are located adjacent to the main rivers in the Netherlands and because they are relatively large so that we can extract a lot of elevation data from them. Square tiles can be extracted from the DTM of these dike rings with elevation data and the entropy of each tile can be computed. The tile with lowest entropy value is indicated with H_{min} and the tile with highest entropy value with H_{max}. For both tiles also the histogram of the elevation is presented. It can indeed be seen that the low entropy tile corresponds to flat terrain and the highest entropy tile to a terrain with a lot of height variation. Also notice that the highest entropy tile is located at the border with Germany, in a region where lot of hills are present.
Figure 4.2 presents the distribution of the entropy of the tiles extracted from the dike rings. This distribution will be used for this input variable in the sampling plan. A random value H_i is chosen with uniform distribution from H_{min} to H_{max} and a tile is chosen with entropy value H closest to H_i.

Another option would be to create DTMs synthetically instead of using 'real' DTM obtained from remote sensing surveys. This can for example be done with the use of random fields. It was however found that generating realistic looking DTMs with line elements such as waterways and roads was challenging, if not impossible, with the use of random field generators.

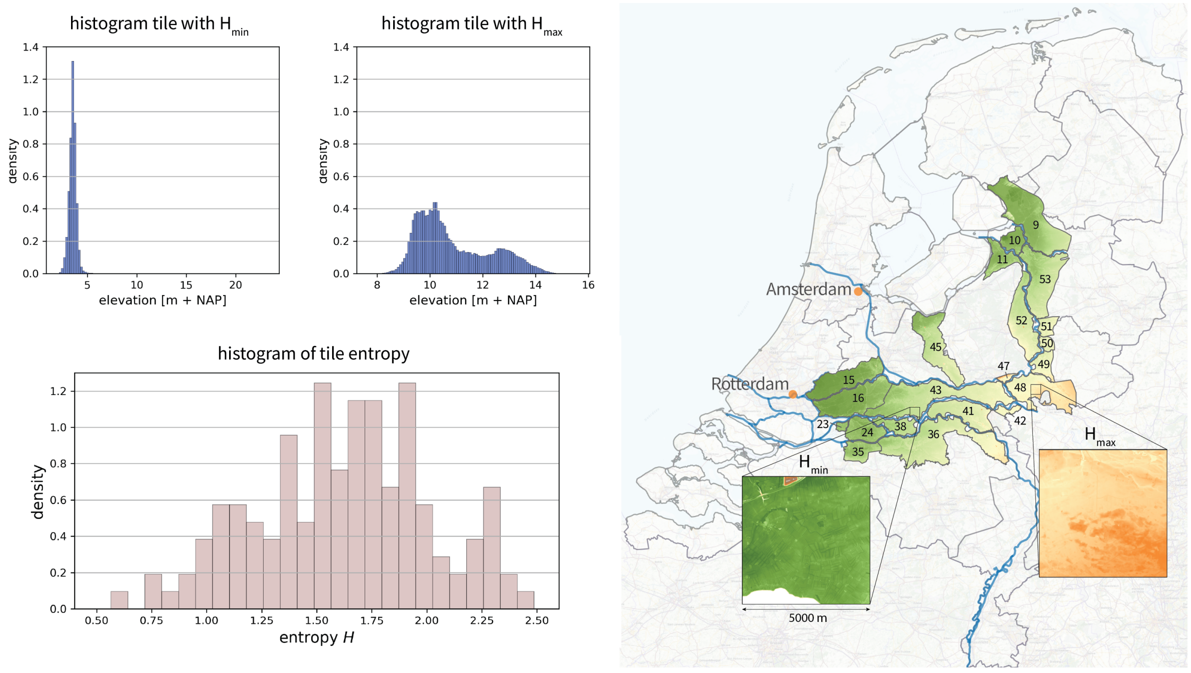


Figure 4.2: **Elevation maps of square tiles extracted from DTM data of major dike rings in the Netherlands**. The dike rings are indicated with numbers, for the names we refer to Table A.2. Histogram of elevation map of the minimum and maximum entropy tiles are presented. The lower left plot shows the distribution of entropy of all elevation tiles used in this study.

Roughness map

Bottom roughness influences the propagation of a flood and can be specified in grid cells of the computational domain. In SOBEK, bed friction can be specified by the equivalent roughness according to Nikuradse. The value of the Chézy coefficient C will then be computed according to the White-Colebrook formula [15].

$$C = 18\log_{10}\left(\frac{12R}{k_n}\right) \tag{4.2}$$

Here, R is the hydraulic radius which is approximately equal to the water depth h and k_n is the Nikuradse equivalent roughness. In this study, we collected land use information of the extracted DTM patches from geo-data files of the Key register Large-scale Topography (Basisregistratie Grootschalige Topografie, BGT, in Dutch). With a conversion table, the land use type can be converted to Nikuradse equivalent roughness value. In this study, we used a standard conversion table from [8]. A small summary of the conversion table used in this study is presented in Table A.1.

Hydrograph boundary condition

Many types of boundary conditions exist and can be used to generate flood simulations. In this study we choose for a simple h-t relation in a single breach grid cell. The breach growth or widening can also be modelled in SOBEK, however in this study we do not take into account breach modelling to reduce the number of input variables. Instead, we use simple triangular hydrographs specified at a single grid cell in the square domain. The hydrograph describes the variation of water depth in the breach cell.

The triangular hydrograph is defined by a duration and a peak which are considered input variables. The duration is a uniformly distributed random variable with minimum of 0 hours and maximum of 24 hours. The peak is also considered to be uniformly distributed with maximum of 6 meters. The latter maximum value is rather large, however not unrealistic for breach scenario's in the Netherlands, where so called polders are sometimes up to 6 meters below sea level. The duration of the simulations is chosen to equal twice the duration of the hydrograph. The maximum duration of a training simulation is thus 48 hours.

The row and column index of the breach location grid cell are also considered input variables. Both are chosen to be uniformly distributed between 400 and 600. We choose this range that to ensure that the breach location is approximately in the center of the computational domain, see 4.3. We want the breach cell to be approximately in the center of the grid to ensure that the boundary of the square domain is not reached by the flood early on in the simulation. When the boundary of the computational domain is reached by the propagating flood, water is reflected. Such reflective straight boundaries are unlikely in application of the surrogate model and therefore may corrupt the training of the surrogate model. The remaining snapshots of the flood simulation are therefore not used in training. The exact time step at which the flood reaches one of the four boundaries of the computational domain is not known a priori and is checked once the simulation is generated.

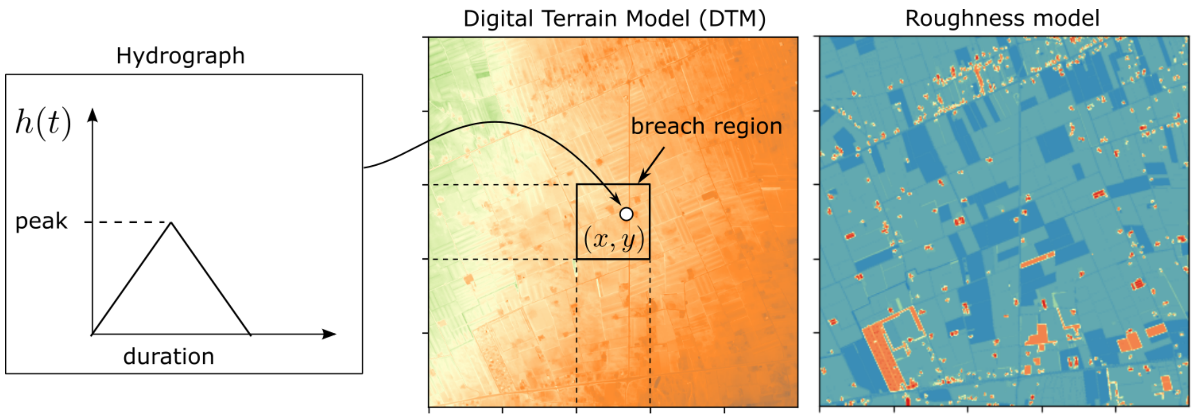


Figure 4.3: **Boundary condition of training simulations**. A triangular hydrograph is defined by a peak and duration as input variables. The breach location in the tile is defined by an x and y location in the inner square as indicated on the left.

4.3.3. Sampling strategy

The choice for the simulations to use for training a surrogate model is important because surrogate models are generally good at interpolation, but have difficulty with extrapolation [10]. A sampling strategy can be used to sample the simulation input denoted with $\mathbf{I} = (I_1, ..., I_m)$ for the m input variables. This input vector determines the output of a single simulation and can be thought of as a point in the input space. The simulation $f(\mathbf{I})$ that results from creating a simulation with a vector in the input space is sometimes called an observation. The set of inputs used for training the surrogate model is called the sampling plan, denoted $\mathbf{S} = \{\mathbf{I^{(1)}}, \mathbf{I^{(2)}}, ..., \mathbf{I^{(n)}}\}$. Often, the number of simulations is limited by the computational budget. A sampling plan should ideally be space-filling [10], such that the surrogate model can be trained to interpolate between the observations. A sampling plan is said to be space-filling if the inputs cover enough range in the input space such that the observations help in efficient training of the surrogate model.

Many sampling strategies exists, examples are pseuro-random sampling, full factorial sampling and stratified sampling. In this framework Latin hypercube sampling (LHS) is used. In LHS the cumulative distribution of the input variables is divided into intervals, which is called stratification. LHS guarantees that samples only appear once for every row and every column. Once a sample is generated, the projection properties along the axes ensure uniform distributions. When one sample is randomly generated, it is often not very space-filling. An optimization method can be used to generate a sample plan with the best space-filling property [36]. This can be done using the distance between the points computed as the p-norm.

$$d(\mathbf{x}_1, \mathbf{x}_2) = \left(\sum_{i=1}^{k} |\mathbf{x}_1^i - \mathbf{x}_2^i|^p \right)^{1/p} \tag{4.3}$$

For which p=1 give the rectangular or Manhattan distance and p=2 gives the Euclidean distance. Here, Ψ_q is a criterion for space-filling properties of a sampling plan $\mathbf{X}$ and was proposed by [26].

$$\Psi_q(\mathbf{X}) = \left(\sum_{j=1}^{,} J_j d_j^{-q} \right)^{1/p} \tag{4.4}$$

In this study, we use five input variables that are used to create a training flood simulation. The input variables are summarized in Table 4.1. A total of 100 samples are created, that can be used to generate 100 flood simulations. A maximin optimization scheme was used with $1e6$ iterations to obtain a sampling plan that is space-filling. In this way, we ensure that we create flood simulations with different inputs and thus aim to maximize the efficiency of training.

Table 4.1: **Description of the input vector $\mathbf{I} = (I_1, ..., I_5)$.** This vector is used to create a flood simulation that serves a training data.

Input	Description
I_1	Information entropy H of the square tile elevation map.
I_2	Peak of the triangular hydrograph.
I_3	Duration of the triangular hydrograph.
I_4	The ith index of the breach cell.
I_5	The jth index of the breach cell.

The created two-dimensional (2D) flood simulations result in a large data set of water depth and flow velocity output files. These flow characteristics can be stored as array data with three channels. The dimension of these arrays are thus 1000x1000x3 for every time step of the simulation. The first channel of this three-dimensional array corresponds to the water depth, the second and third channels contain the flow velocities in x- and y direction.

4.4. Patch-wise prediction

The output of a flood simulations is stored as three-dimensional array data consisting of flow snapshots over a regular grid of size $H \times W$, where H and W denote the number of grid cells in the x- and y direction of the 2D domain. Often high elevation contours restrict the size of a flood prone area. Outside the flood prone area domain, grid cells are assigned no-data values and no numerical computations are made in these grid cells. In SOBEK, a rectangular 2D grid for a certain dike ring covers the flood prone area and can be large or small depending on the size of the dike ring and the grid resolution.

In this study we will use patches, that are smaller pieces of a spatial domain, in training and application of the surrogate model. Patch-wise prediction is used because it allows the surrogate to focus only on regions where the flood has reached and not on regions where no flood occurs. Figure 4.4 shows how the surrogate model uses patch-level predictions to create a surrogate simulation of a flood in a dike ring area. It can be seen that only predictions are needed for those patches that contain part of the flood extent. Trained CNNs will be used to make patch-level predictions of the flow characteristics.

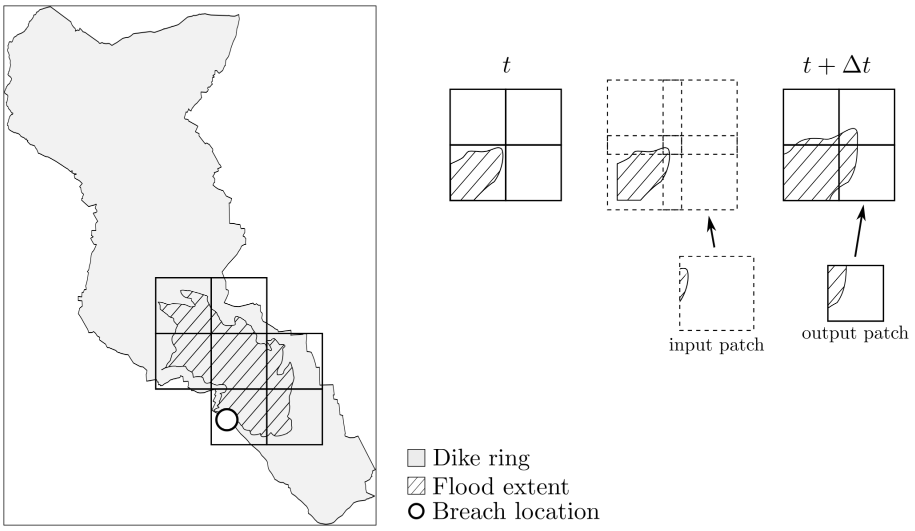


Figure 4.4: **Proposed patch-based prediction method.** By using patches, we force the surrogate model to focus on regions where the flood has reached. Input patches have larger dimension and contain part of the neighbouring output patches to give the surrogate spatio-temporal context.

A possible drawback of the patch-wise prediction approach is that the trained surrogate will have to make multiple prediction, instead of one prediction for the larger domain. Making one prediction for each patch may be less efficient. In addition, the predictions of flood propagation in individual patches have to be connected again. With connecting the patch predictions we face the problem that the propagation of the flood in one patch influence solutions in other patches. This problem is also encountered in parallel computation algorithms applied to flood simulations. One can imagine that patches should be given a certain context of the surrounding patches in order to make accurate predictions. Within one time step Δt, the flood may propagate from one patch to the next.

To give the surrogate model spatio-temporal context, we use input patches that cover a larger part of the spatial domain than the output patches. In Figure 4.4, it can be seen the the grid of input patches is partly overlapping. This means that the input of the neural network contains a small part of the neighbouring patches. Output patch predictions are placed exactly next to each other without any overlap to reconstruct a flood snapshot at a certain time t. If a flood is about to reach the boundary of an output patch, it is detected in the input of a neighbouring patch. Therefore it should be possible to have flood propagation between patches.

4.5. Patch-wise training

For using the surrogate model to make patch-wise predictions, it is not strictly required to train on patches as well. Once trained, the convolutional neural networks are not restricted to the images sizes encountered during training but can be applied to inputs of arbitrary size. However, we used patch-wise training strategy because the flood simulation output of size $H \times W = 1000 \times 1000$ was found to be large and thus taking up much of the memory of a graphical processing unit (GPU). Large arrays may not fit in GPU memory or it can pose restrictions on the batch size used during training. For the same reason it may also pose restrictions on the network size, that is the number of (convolutional) layers and filters. A patch-wise training strategy is also sometimes used as a data-augmentation step because with overlapping patches the number of training examples can be increased. In this study, the output patches extracted from the flood simulations are not chosen to be overlapping for simplicity reasons. This opportunity for data-augmentation thus was not utilized.

4.5.1. Patch dimensions

For the dimensions of the input patches we choose 232×232 grid cells and for the output patches we choose 128×128 grid cells. With a larger patch dimension, you need less patches to cover a dike ring area and therefore the surrogate also needs to make less predictions to emulate the flood simulation. We choose these dimension because they are common image sizes used in training deep learning models. The size difference between in- and output patches equals 52 on all sides. This corresponds to 260 meters with the mesh size $\Delta x = 5$ meter. With a time step of $\Delta t = 1$ minute, we can calculate the flow velocity limit below which the propagation of the flood between two neighbouring patches can be predicted $u_{limit} = 4.33 m/s$. Above this limit, the flood propagates too fast and a smaller time step may be used for more accurate prediction.

4.5.2. Patch extraction

To create training examples for deep learning architectures, we extract couples of input-output of which the input will be the flow characteristics at time t as well as elevation and roughness maps for a given patch. The network output will be the flow characteristics at a given time step Δt later for the same patch. Figure 4.5 shows how these patches are extracted from the snapshots of a given flood simulation. The dimensions of the in- and output patches and of the flood simulation grid were chosen such that exactly 7 output patches fit inside a row or column of the simulation grid. For extracting patches, it must be known at what time step water is entering an output patch. For this, arrival times are computed and stored in separately.

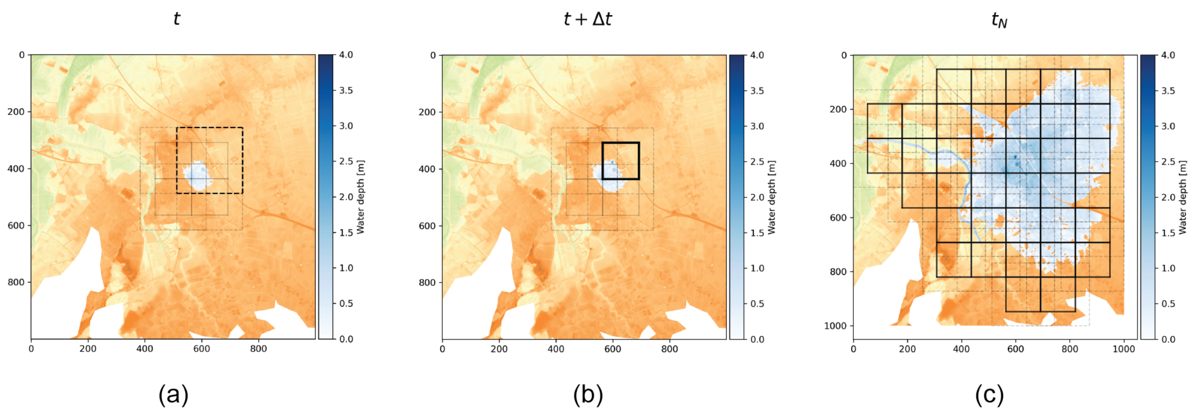


Figure 4.5: **Extraction of input- and output patches from a training simulation snapshots**. **(a)** Extraction of an input patch with water depths at time step t. **(b)** Extraction of an output patch with water depths at time step $t + \Delta t$. **(c)** In- and output water depth patches at time step N_{bound}, when water hits one of the domain boundaries. Input patches are indicated with a dotted line and are larger than the output patches indicated with a solid line.

A final step is to track down the time step at which water hits the boundary of the computation grid, indicated with N_{bound}. The snapshots after this time are not taken as training data as they are influenced by the reflective straight boundary. This corrupts the training data as such boundaries are not likely to be present in practical applications of the surrogate.

4.5.3. Pre-processing steps and data augmentation

As a last step the training examples are processed to make get the best training results for the deep learning architectures. In grid cells of the output of flood simulations generated with SOBEK where no terrain information is present, that are cells outside of the valid computational domain, are returned as no-data values. In SOBEK, the default is -999 for terrain and -999.999 for flood characteristics. These default values lie well outside the observed range for the characteristics that one expects during a flood. These no-data values can cause difficulties during training and for that reason they are changed by zeros.

The elevation maps from LiDAR surveys are given with respect to NAP, and used for simulations for SOBEK as such. To make training easier, the datum was adjusted in the training set to the elevation at the breach location. This ensures that the elevation values in the training set are approximately in the same range. The friction values are all between zero and ten, so no adjustments are made for this input.

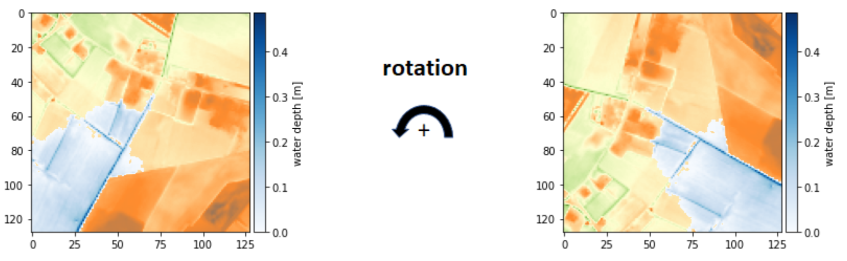


Figure 4.6: **Data augmentation step**. Input and output examples are randomly rotated with angle $\theta = [0°, 90°, 180°, 270°]$.

With deep learning in general it is often a good idea to perform data augmentation steps. With flood inundation simulations, a possible augmentation step is to use 90 degree rotations of the patches. In the training phase, this random augmentation step is used here. Figure 4.6 present the augmentation of a patch.

4.6. Deep learning architectures for the surrogate model

In this section, three deep learning architectures are proposed for the use in the surrogate modelling framework. These architectures will be trained on patch-level examples and tested in a surrogate model to emulate 2D flood simulations. We discuss the choice for the input and output of these networks. After that we describe the layout of two feed-forward architecture and one recurrent architecture.

4.6.1. Network in- and outputs

As discussed in Chapter 2, 2D flood simulation snapshots ϕ_t have grid-like structure similar to images. To emulate flood simulations, we use an image-to-image regression network that can be used to predict future snapshots ϕ_{t+1} given input images. For patch-level inputs of these networks we use flow characteristics at past time steps as well as the elevation and roughness maps. To create a spatio-temporal learning setting, the output of the network has a certain time shift compared to the input. The output of a network may be the flood characteristics shifted with one time step Δt. Since the DTM and roughness are considered not to change in time, they are not included in the output of the network.

4.6.2. Convolutional encoder-decoder network

The first network architecture that we will train and test is illustrated in Figure 4.7. This is a feed-forward architecture with an encoder-decoder structure and convolutional layers such that it can take a multi-channel image and input and produce an output of similar data type. The encoder of the network architecture uses pooling layers to decrease the dimension of the input. In doing so, convolutional layers are used to extract feature maps f. Here only convolutional layers are used to introduce trainable weights and no fully connected layers are used in the information bottleneck. We did not choose to use fully connected layers as this greatly increases the number of weights of the network. In the encoder part of the network, max pooling layers are present. It can be seen that the number of feature maps f is increased by using the convolution layers. In this way, the loss of information due the pooling operations is reduced. In the last layer of the network, the output is cropped to the output patch size of 128×128. In the decoder part of the network, upsampling layers are present that use linear-interpolation to increase the dimension of the input. As a last layer, the output can be cropped to a different width and height. In this way, we can make the output patches of smaller size than input patches. In addition, the cropping layers removes possible boundary artifacts that are sometimes present in the predictions because of the convolutional operations [27].

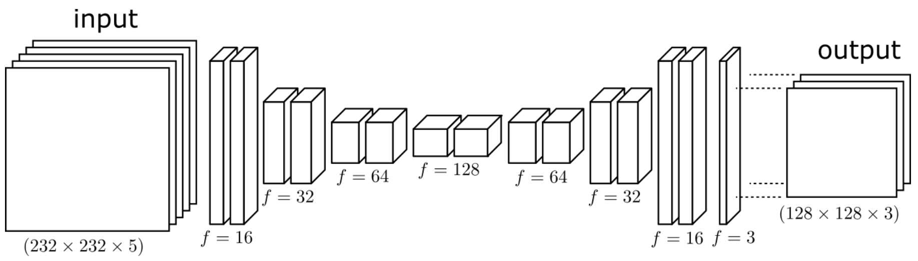


Figure 4.7: **Network layout of the convolutional autoencoder architecture.**

The input of the convolutional autoencoder network consists of an image with 5 channels. The first three channels contain the water depth h_t, flow velocity in x-direction u_t and flow velocity in y-direction v_t in a certain patch at a time t. The last two channels include the elevation and roughness maps of the patch. The output of the network consists of the flow characteristics h_{t+1}, u_{t+1} and v_{t+1} at a time step $t + \Delta t$.

4.6.3. Modified U-Net

The second feed-forward network that we have trained is a modified U-Net architecture, the layout of this network is presented in Figure 4.8. Three skip connections are added to the architecture where filter maps are copied and concatenated with output of convolutional layers deeper in the network.

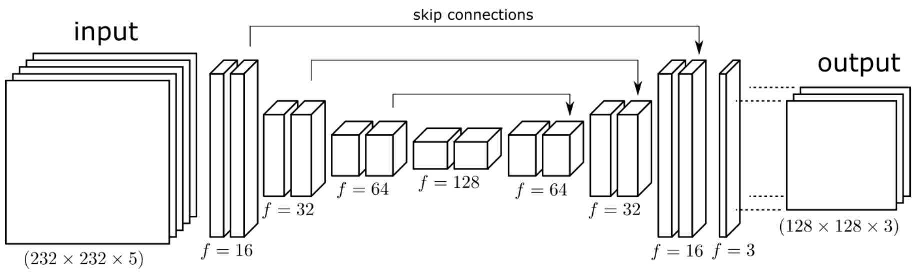


Figure 4.8: **Network layout of the the U-net architecture.**

These skip connections avoid the loss of information caused by pooling and upsampling. This can be very effective as the input- and output of the network are flow snapshots that are shifted by a small time step and thus in most cases look very similar. The last convolutional layer of this network produces three feature maps and which are cropped to get the desired output size of 128 × 128 × 3 of the output patch.

4.6.4. Convolutional LSTM network

The final architecture that is proposed is a recurrent architecture that has an encoder for the flow characteristics input sequence and a separate encoder for the elevation and roughness map. The reason for choosing a separate flow encoders and terrain encoder is that the terrain data is static and it would be redundant to add to each of the elements in the sequence. This is undesirable as the image in the sequence lead to high memory consumption during training as well as in application of the surrogate. The latent space of the flow encoder and the terrain encoder are later concatenated and after that a decoder reconstructs the output. For the flow encoder, an input sequence $X_{flow} = [\phi_{t-3}, \phi_{t-2}, \phi_{t-1}, \phi_t]$ is given with dimension 4 × 232 × 232 × 3. Although a single flow characteristics $\phi_t = (h_t, u_t, v_t)$ would contain enough information to predict future frames, assuming the boundary condition is known, it may be easier to train a network with a larger sequence and it may result in better performance since more information is available. Figure 4.9 presents a schematization of the network layout. A flow encoder network operates on the input sequence and two convolutional LSTM layers are used. The first convolutional LSTM layer outputs a sequence and the second gives a single output with feature maps. These are added to the feature maps resulting from the terrain encoder. After concatenating, there are fed into the encoder network that generates an output patch of dimension 128 × 128 × 3.

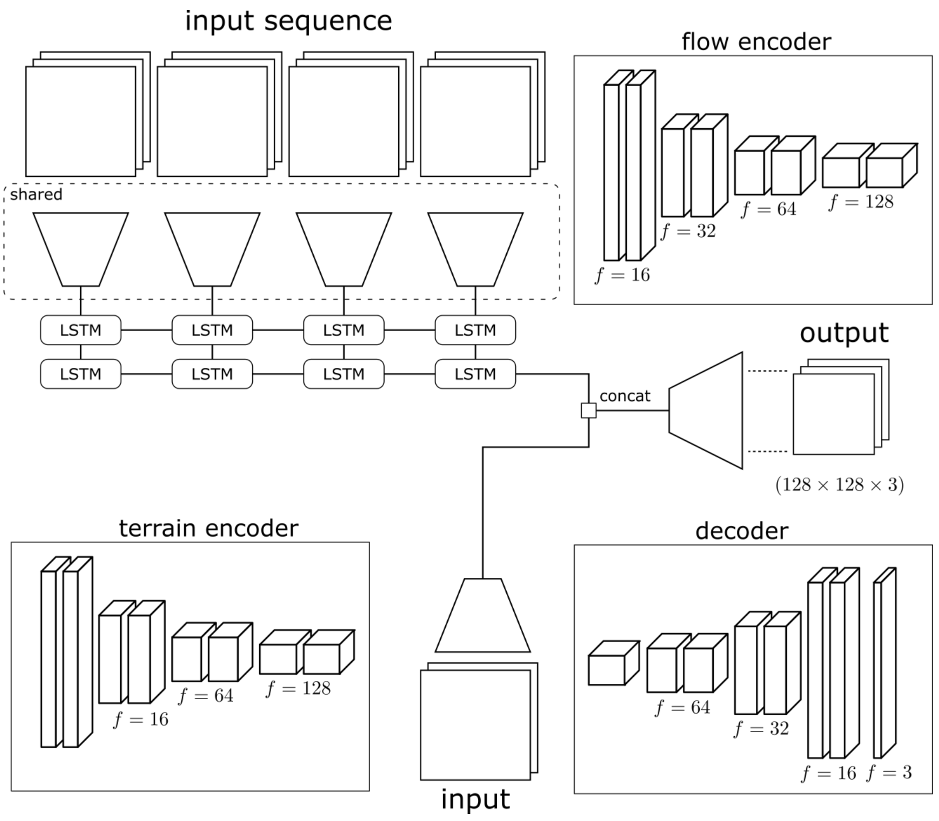


Figure 4.9: **Network layout of the ConvLSTM network.** This architecture has a separate flow encoder and terrain encoder that take as input a sequence of past flow frames and the patch terrain maps respectively.

5 Network training

In this Chapter we discuss the training process of the network architectures proposed in Chapter 4. We discuss how the simulation data set will be divided into a training set, validation set and test set. In addition, the different training runs which include variations in the used loss function, the time step between examples and the use of different input-output type are summarized. We discuss the choice for the hyperparameters that were used for the various training runs. The training time of the different architectures is is also presented and we present the test simulation that will be used to test the various trained architectures.

5.1. Training set

As discussed in Chapter 4, a total of 100 two-dimensional (2D) flood simulations were created with the software program SOBEK and the output of these simulations was processed further to create a large set of input and output patches containing flow characteristics and terrain data. We use the full set of simulations and divided these randomly in a training, validation and test set. The patch examples in these sets are randomly shuffled to avoid overfitting. Patch examples from training, validation or test set are not mixed with one of the other sets. The statistics of the data sets resulting from the random split are presented in Table 5.1. Here N is the number of time steps in the simulation and N_{bound} is the number of time steps until water hits one of the domain boundaries.

5.1.1. Training scenarios

We found that the training data set was large which leads to long training times of the network architectures. Because of limited time, a second training scenario is defined which contains a fraction of the full data. In this partial training scenario's, 15 training simulations and three validation simulations are randomly chosen from the set of training simulations in the full training scenario. The statistics of this partial training scenario data set are also given in Table 5.1. It can be seen that the data in the partial training scenario is similar to that of the full training scenario, such that results for architectures on this smaller data set may also mean that it performs good in full training.

It can be seen from Table 5.1 that not exactly all simulations are used in the full training scenario. Two simulations were left out because the boundary conditions of these simulations was found to be such that no simulation was generated. In one of the simulations, the hydrograph peak was very close to zero and the other simulation had very short duration. SOBEK often does not produce output for numerical computational reasons when the boundary has very small water level increments [15].

Table 5.2 presents some statistics of the run times of simulations created in SOBEK. The simulation with the longest calculation time took around 13 days to finish on a CPU core. Because the duration of the simulations varied due to input sampling some simulations were rather fast as they only consisted of a couple of frames. On average, the simulations took around 2 days to complete. We also show the calculation time until N_{bound}, which indicates the duration of the simulations that are valid as training examples. That is, the run time of the simulations when we remove the frames after the moment that water reach one of the domain boundaries. It

	Partial training		Full training		
Property	**Training set**	**Validation set**	**Training set**	**Validation set**	**Test set**
Simulations	15	3	92	4	2
Examples	84,325	23,518	479,268	26,514	9,868
Percentage of total	78.19	21.81	92.94	5.14	1.91
Mean tile entropy	1.51	1.50	1.54	1.37	1.19
std tile entropy	0.64	0.51	0.56	0.50	0.45
Mean peak	3.43	2.91	3.00	2.65	1.11
std peak	1.94	1.88	1.74	1.69	0.71
Mean duration	13.92	18.89	11.83	16.23	13.34
std duration	6.78	5.24	7.01	6.47	3.98
Mean N	794.13	1314.67	679.82	1230.00	861.00
std N	345.01	163.24	408.51	203.69	241.00
Mean N_{bound}	417.07	678.33	363.48	569.75	468.00
std N_{bound}	380.39	546.25	288.47	509.08	152.00

Table 5.1: **Statistics of the training, validation sets**. A partial training scenario and a full training scenario are defined.

can be seen that the maximum calculation time in this case is much shorter. This indicates that the duration of some of the simulations was chosen too large and a lot of the data resulting from these simulations is not being used.

	N_{bound}	N
Mean run time	0 days 11:38:43 hours	1 days 21:51:20 hours
Maximum run time	1 days 11:12:39 hours	13 days 00:06:07 hours
Minimum run time	0 days 22:51:20 hours	0 days 00:22:06 hours

Table 5.2: **SOBEK simulation run times**. Average values of 100 flood simulations on square tiles of 1000 × 1000 grid cells.

5.1.2. Training runs

Various training runs are performed for the different network architectures. For the convolutional encoder-decoder network, two training runs were performed. In the first run, the input X contains the flow snapshot $\phi_t = (h_t, u_t, v_t)$ and terrain maps of a certain input patch and the output Y contains the flow snapshot $\phi_{t+1} = (h_{t+1}, u_{t+1}, v_{t+1})$ at time step $\Delta t = 1\ min$ later. In the second training run, the output is changed to the discrepancy between two successive time steps, that is $Y = \Delta\phi = \phi_{t+1} - \phi_t$. In both runs, a MSE loss function was used and a time step of $\Delta t = 1\ min$. Table 5.3 summarizes the settings of the two training runs.

run	**time step**	**input size**	**output size**	**output**	**Loss**	**scenario**
1	1 min	$(232, 232, 5)$	$(128, 128, 3)$	$\phi_t = (h_{t+1}, u_{t+1}, v_{t+1})$	MSE	Partial
2	1 min	$(232, 232, 5)$	$(128, 128, 3)$	$\Delta\phi = \phi_{t+1} - \phi_t$	MSE	Partial

Table 5.3: **Training runs performed for the convolutional encoder-decoder network**.

Figure 5.1 presents the loss during training as well as the validation loss evaluated at the end of each epoch. It can be seen that the validation loss is lower than the training loss throughout the epochs. A reason for this can be that the validation set contains simulations that are easier for the model to predict accurately. Another explanation is that the validation loss is evaluated after the epoch whereas the training loss is computed during the epoch in between batches.

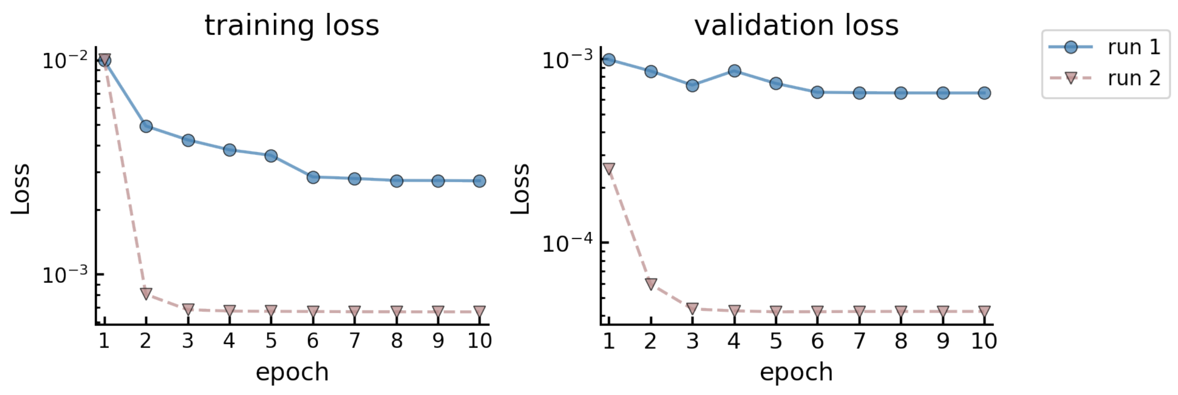


Figure 5.1: **Loss during training and on the validation set for various training runs of the convolutional encoder-decoder network**.

For the modified U-net architecture, a total of four training runs were performed. In the first and second runs, the network was trained to predict ϕ_{t+1} for time step $\Delta t = 1\ min$ and $\Delta t = 2\ min$ respectively. In the third training run, the outputs were the discrepancies between the current and next time step $\phi_{t+\Delta t}$ with time step $\Delta t = 1\ min$. For all three above described training runs a MSE loss was used. Finally, in the fourth training run a physics informed loss function was used. Table 5.4 summarizes the settings of the four training runs.

run	time step	input size	output size	output	Loss	scenario
1	1 min	(232, 232, 5)	(128, 128, 3)	$\phi_t = (h_{t+1}, u_{t+1}, v_{t+1})$	MSE	Partial
2	2 min	(232, 232, 5)	(128, 128, 3)	$\phi_t = (h_{t+1}, u_{t+1}, v_{t+1})$	MSE	Partial
3	1 min	(232, 232, 5)	(128, 128, 3)	$\phi_t = (h_{t+1}, u_{t+1}, v_{t+1})$	PI	Partial
4	1 min	(232, 232, 5)	(128, 128, 3)	$\Delta\phi = \phi_{t+1} - \phi_t$	MSE	Partial

Table 5.4: **Training runs performed for the modified U-Net architecture**.

Figure 5.2 presents the loss during training as well as the validation loss evaluated at the end of each epoch.

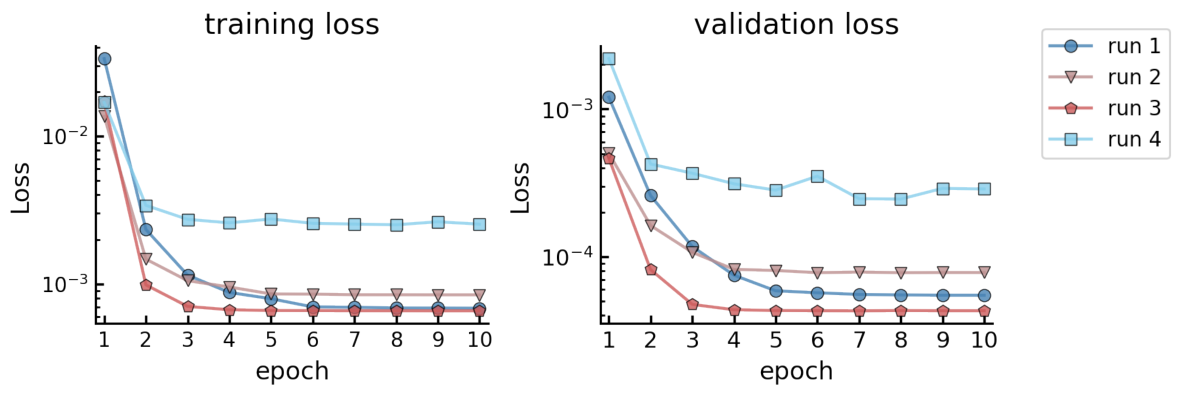


Figure 5.2: **Loss during training and on the validation set for various training runs of the modified U-Net network**.

Finally the ConvLSTM network was trained on the partial training set. The network was trained once on predicting discrepancies $\Delta\phi$

run	time step	input size	output size	output	Loss	scenario
1	1 min	$(4,232,232,5)$	$(128,128,3)$	$\Delta\phi = \phi_{t+1} - \phi_t$	MSE	Partial

Table 5.5: **Specifications of the training run performed with the ConvLSTM architecture**.

5.2. Hyperparameter choice

In this section we discuss the choices for the hyperparameters used in training the network architectures. These parameters must be specified before training starts and influence the training process. Ideally, the influence of hyperparameters choices should be explored and an optimization strategy can be used to find optimal parameters. However, due to long training time, in this study the influence of hyperparameters was not explored. The hyperparameters are based on computer specifications and on values used in examples such as in [7] and literature.

A first hyperparameter choice that will be discussed is called the batch size. Deep learning models are trained on training examples, however generally a batch of examples is created after which an update is made to the trainable weights. There are two reasons to use batches instead of training on the whole data set at once [30]. Firstly, when a training data set is large it may not be possible to load the whole data set at once and you are forced to train on batches of the data set. Secondly, training on smaller batches speeds-up training as you are updating the weights after each batch. When small batches are used, the number of updates per epoch increases. Only making an update of the weights after going through all training examples may not be efficient. A disadvantage of using small batch sizes is that the accuracy of the gradient will be smaller. The size of the batches is usually constrained to memory of the graphical processing unit (GPU). A typical value is 32, 64, 128 etc. Often powers of 2 are used because operation can work faster when their inputs are sized in powers of 2. Here a batch size of 16 was chosen because it is used very often in training deep learning models and because it was possible within memory constraints of the GPU used in training.

A second hyperparameter is the learning rate of the optimizer which determines by how much the weights are adjusted after each training step. Choosing an appropriate right learning rate can be challenging. When a small value for the learning rate is chosen, the convergence of training may be slow. When a very large value is chosen, the network may not converge to a minimum of the training loss [30]. The learning rate of the model is sometimes seen as the hyperparameter that has largest influence on the performance of the model. Here, a learning rate of $1e-4$ was used. Also, the learning rate was decreased as soon as the validation loss stops improving.

A third hyperparameter is the actual optimization method used for training the deep learning model. Here Adaptive Moment Estimation (Adam [19]) is used as an optimization algorithm. This optimizer is available to use in many deep learning libraries and it has been showed to give good performance for many learning tasks.

In addition to the above choices for hyperparameters, the length of training was chosen to be 10 epochs. That means that the networks goes through all training examples 10 times. The loss function that is used in training is the mean squared error (MSE). During training also the mean absolute error (MAE) is tracked. The choices for hyperparameters are summarized in Table 5.6.

5.3. Training time

We found that training the network architectures on the partial data set with training 10 epochs took approximately 1.5 days for the convolutional encoder-decoder network and modified U-Net, which was feasible for trying different training runs in the available time for this research. Estimates of the network training times are summarized in Table 5.7. Training was done on a 8 GB Nvidia GeForce RT2080 GPU in all cases. It was not clear how much of the capacity of this GPU was used during training. The efficiency of the training process may be improved with a more efficient input data pipeline which was not explored in this study. The training time of the ConvLSTM network was found to be significantly longer than the training time of the other models. This can be explained by the network complexity as the recurrent layers in the ConvLSTM network increase the number of weights of this network architecture. Also, the input of these network is four

Table 5.6: **Hyperparameters used in training runs**. For the learning rate, a plateau was used such that the learning rate is decreased when validation loss starts to increased.

Parameter	**Value**
Batch size	16
Learning rate λ	0.0001
Pptimizer	Adam
Loss	MSE
Metric	MAE
Epochs	10

times larger than the inputs of the non-recurrent network. Loading these inputs during training therefore also takes more time which leads to a longer overall duration of training. One training run was performed using the full training set, however the results were not included in this report as due to an error in the loss function which corrupted the results of the trained network. The training time of this run was approximately 9 days, which is significantly longer than the training time of the partial data set. The larger validation set of full training was also larger, which contributed to the longer duration as well. The ConvLSTM network was not trained on the full training set because of time limitation of this research.

Table 5.7: **Training time of different network architectures**. These are estimates for 10 epochs using the partial data set.

Architecture	Partial training set	Full training set
CAE	$\approx$ 1.5 days	$\approx$ 9 days
U-Net	$\approx$ 1.5 days	$\approx$ 9 days
ConvLSTM	$\approx$ 5 days	-

5.4. Test simulation

In this section we discuss one of the simulations in the test set which will be used in Chapter 6 to test the performance of the surrogate model with various trained network architectures. Figure 5.3 gives the initial- and boundary conditions of the test simulation. From the elevation and roughness maps, it can be seen that the terrain is mostly rural area. The friction coefficients are low for most of the area indication grass or agricultural area with only a few buildings present along a main road. The hydrograph is presented in Figure 5.3 (c) and shows that the water rises to about 1.7 meters in 300 minutes. Shortly after the peak is reached, water hits one of the boundaries of the domain and the rest of the simulations neglected.

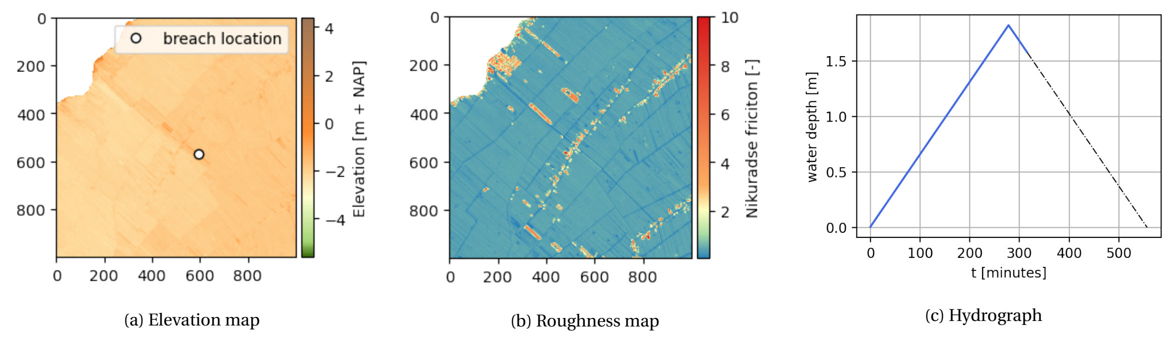


Figure 5.3: **Initial- boundary conditions of simulation 23 from the test set**. (a) The elevation map of the domain and the breach location, (b) the roughness map and (c) the triangular hydrograph specified in the breach location.

To see how the surrogate performs on the described test simulation with different starting points, we use four starting times $t_0 \in [0, 10, 50, 100]$ minutes. The water depth and flow velocities at these time instances are presented in Figure 5.4. It is expected to be more challenging for the surrogate model to predict the flood propagation for start time $t_0 = 0$ minutes because the amount of information in the initial conditions is smaller than for the later start times. The information provided to the surrogate model comes solely from the water level variation in the breach cell for starting time $t_0 = 0$ minutes. The flood has to be predicted completely from this information. With $t_0 = 100$ minutes, much more of the computational domain is flooded and the amount of information that is provided to the surrogate is much larger.

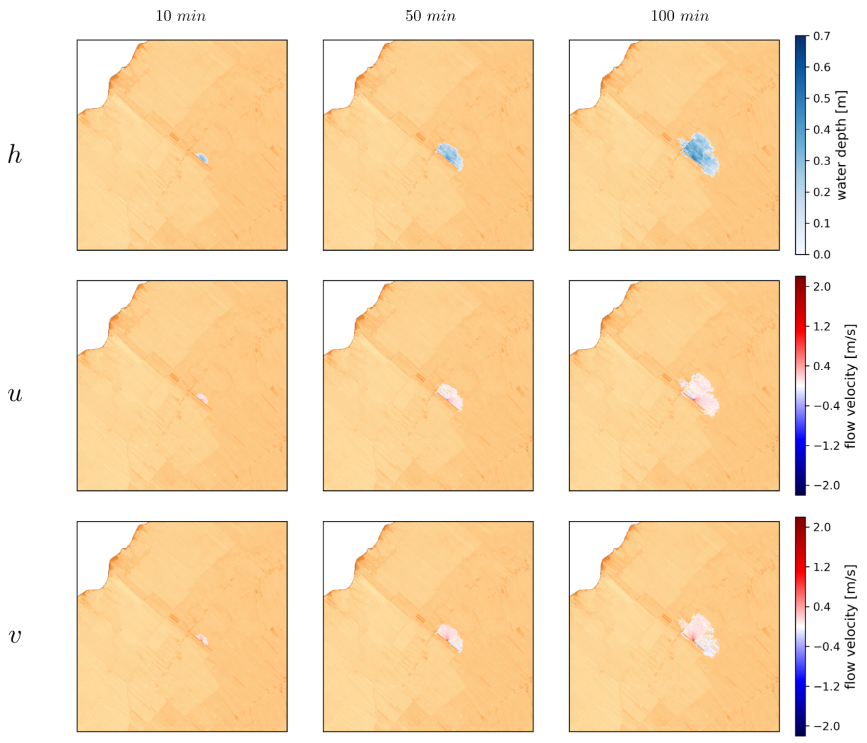
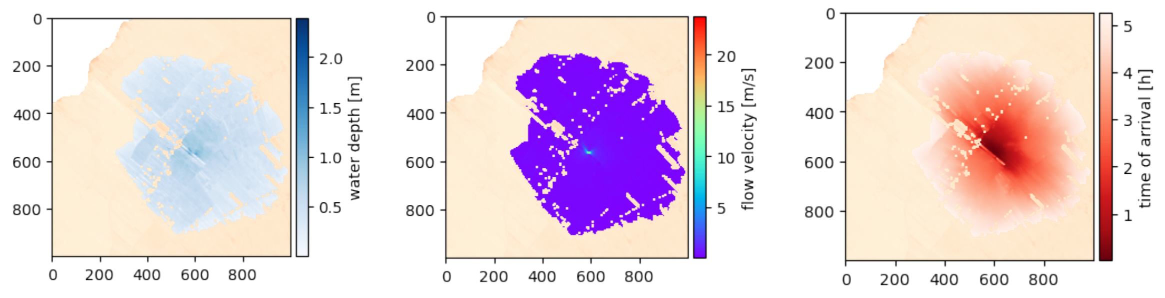


Figure 5.4: **Initial conditions at different starting times t_0.**

Figure 5.5 presents the inundation maps of the test simulation, it can be seen that the flood extents more towards the rural area away from the buildings in the small town. Near the breach, high flow velocities are observed with an extreme of more than 15 m/s. Away from the breach, much lower maximum flow velocities occur. Water depths in the overall flood stay below 2 meters in most of the flooded area.



(a) **Maximum water depth** (b) **Maximum velocity** (c) **Time of arrival**

Figure 5.5: **Ground truth inundation maps for the test simulation**.

6

Feed-forward architecture

In this chapter, we discuss the results of the trained feed-forward architectures. These are a convolutional encoder-decoder network and a modified U-Net architecture. We use the trained networks resulting from the different training runs as described in Chapter 5 to create surrogate simulations and emulate the test simulation. The performance of the surrogate model with the trained networks is then compared to the test simulation generated with SOBEK.

6.1. Convolutional encoder-decoder network

Here we discuss the performance of the convolutional encoder-decoder network as presented in Figure 4.7. The network is trained with the settings presented in Table 5.3.

6.1.1. Test set performance

The convolutional encoder-decoder architecture was first trained on the partial training set with a MSE loss function and 10 training epochs. Figure 6.1 presents the Root Mean Squared Error (RMSE) in the water depth h and flow velocity $\mathbf{u} = \sqrt{u^2 + v^2}$ for the surrogate simulation for different starting times $t_0\ in[0, 10, 50, 100]$ minutes. With begin time $t_0 = 100$ minutes, the flood has propagated over a region of the domain and thus there is more information in the input patches of the surrogate model. Therefore, we expect that it may be easier for the model to predict future time steps of the simulation than for starting time $t_0 = 0$. It can be seen that due to the error propagation, the RMSE increases with the number of time steps of the simulation. The RMSE in h remains within 10 cm for about 22 time steps and does not show a dependence on the begin time t_0 of the simulation. In the flow velocity $\mathbf{u}$, it can be seen that the RMSE increases more rapidly for the larger begin time of $t_0 = 100$ minutes.

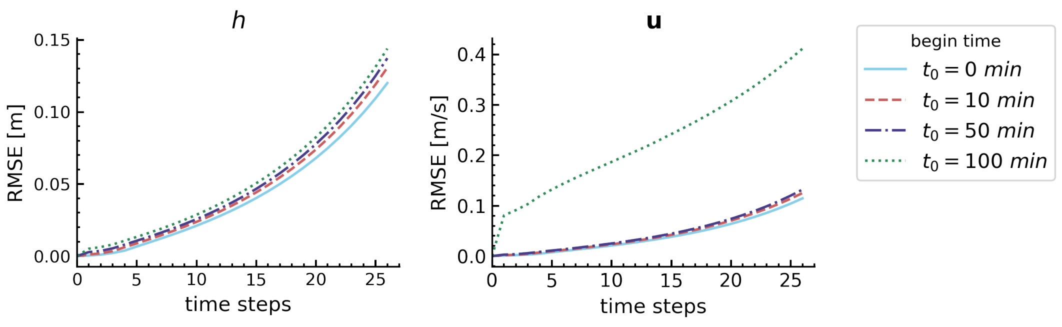


Figure 6.1: **Test simulation RMSE for convolutional encoder-decoder network**. The error in water depth h and flow velocity $\mathbf{u} = \sqrt{u^2 + v^2}$ is presented for different begin times t_0.

6.1.2. Error reduction with threshold filter

From Figure 6.2, it can be seen that small errors are introduced in the predictions of the trained network at each time step. These small errors start to rapidly spread over the flood domain since they also occur at the boundary of the patches. These water depth errors will quickly appear in the overlap regions between two patches which causes a propagation to neighbouring patches. After four time steps of $\Delta t = 1$ min, these errors already reached the boundary of the domain. In this section, we investigate post-processing steps that can be used to reduce the rapid propagation of small errors over the flood domain.

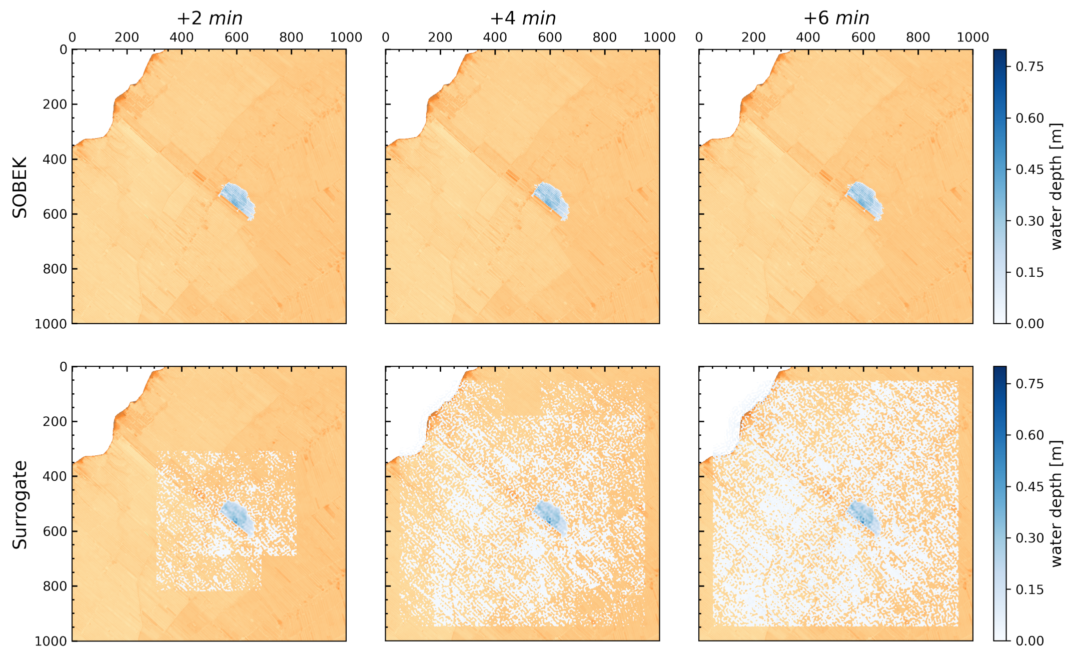


Figure 6.2: **Water depth predictions for the convolutional encoder-decoder network**. Here, snapshots of the ground truth simulation (SOBEK) and the surrogate simulation are presented.

The errors propagate to neighbouring patches quickly because at each time step the surrogate checks if there is water present in each input patch of the grid. From the results, it can be seen that at every time step, the flow prediction contains small errors over the entire area of the patch. These small errors in the overlap regions triggers the surrogate to make predictions for a neighbour patch. To reduce the propagation speed, a water depth threshold $\theta_h > 0$ can be defined such that neighbouring patches only start making flow predictions once the water depth in these patches exceed this threshold value. In other words, at each time step we check for each input patch if there is at least one cell with water depth above this threshold. If that is the case, the trained network is used to make a flow prediction for these patches. Another approach to reduce the propagation of these errors is to use a filter. That is, water depths below a certain threshold value are removed from the flow predictions.

In Figure 6.3, We used the same trained network to create surrogate simulations for start time $t_0 = 10$ min. We used a threshold filter and an approach were we delay the propagation of errors and compared these to a surrogate simulation without error reduction. It can be seen that a filter to patch predictions and removing small water depths below a threshold θ_h is most effective in reducing of the RMSE in water depth h and flow velocity $\mathbf{u}$. For this error filter, a threshold value of $\theta_h = 0.01$ m was used. In the other approach, we check if water depths in the input patches exceed a threshold $\theta_h = 0.01$ m. If that is the case, the patches that share the overlap region will start to make predictions in the future time step. However, if the water depths in the overlap regions are smaller than the threshold value the predictions are not induced. In this way, we can delay the propagation of the error over the computational domain. It can be seen that this approach however leads to a smaller reduction of the RMSE compared to the error filter approach. For this reason we will use an error filter for the surrogate simulations in the remaining of this study.

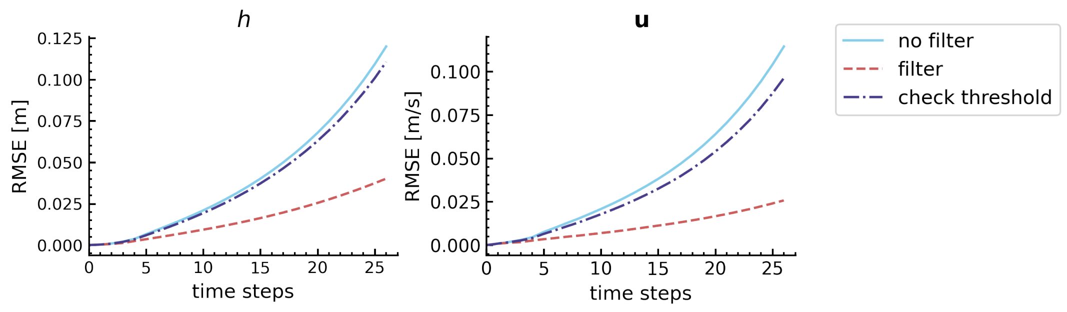


Figure 6.3: **Test simulation RMSE for noise filters**. For begin time $t_0 = 10\ min$ with water depth threshold $\theta = 0.01\ m$.

To investigate the value of the threshold θ_h used in the error filter approach, we create three simulations with $\theta_h \in [0.005, 0.01, 0.05]$ meters for start time $t_0 = 10\ min$. Results of the simulations are presented in Figure 6.4. It can be seen that a larger value of θ_h reduces the RMSE of the water depth h and flow velocity $\mathbf{u}$, however the the flood extent decreases in this case. The flood extent is the percentage of cells in the computational domain that contain water depth $h > 0$. We can see that the SOBEK simulation increases slowly over time. Using a large filter threshold $\theta_h = 0.05\ m$ causes a decrease of the flood extent, indicating that the flood prone area is drying out. The filter threshold of $\theta_h = 0.01\ m$ leads to flooding behaviour which spreads more rapidly than the SOBEK simulation. The RMSE error in water depth is also presented and indicates the error in the area of non-zero water depth h_{wet} in the SOBEK simulation. Here, a larger value of θ_h increases the RMSE in h_{wet}. From these four plots, it can be seen that using an error threshold of $\theta_h = 0.01\ m$ gives results that are closest to the SOBEK simulations which we consider as ground truth.

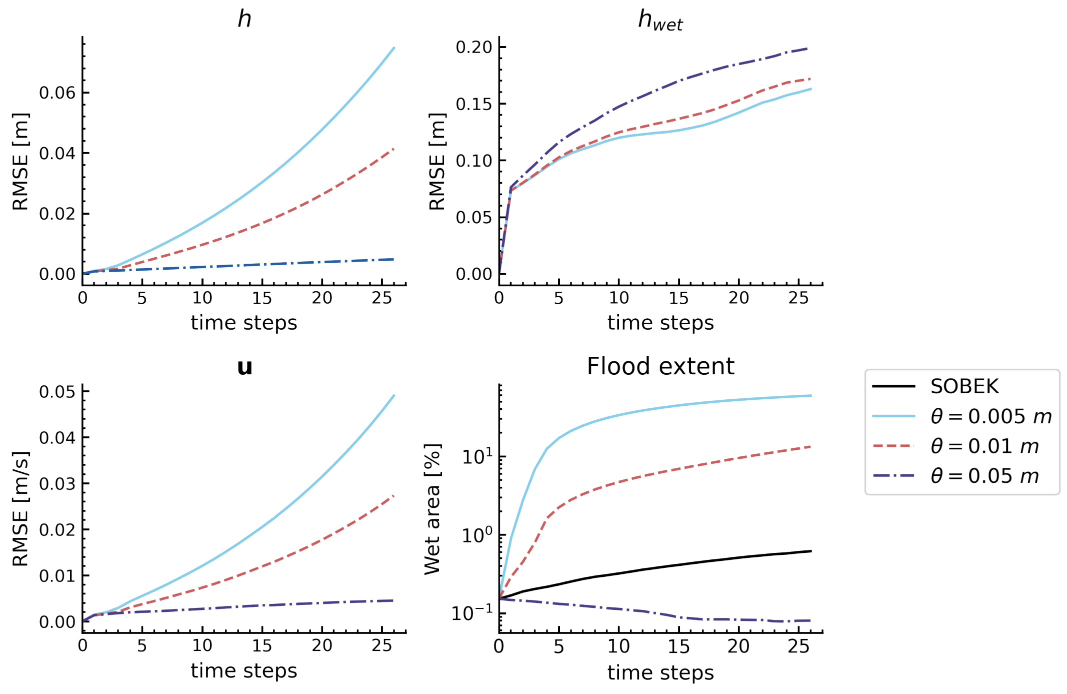


Figure 6.4: **Test simulation RMSE for threshold values of water depth**.

6.2. Modified U-Net

Here we discuss the performance of the modified U-Net architecture as presented in Figure 4.8. The network is trained with the settings presented in Table 5.4. We also compare the modified U-Net with the convolutional encoder-decoder model.

6.2.1. Test set performance

Figure 6.5 presents the performance of this modified U-Net architecture on the test simulation with begin time $t_0 = 10\ min$ compared to the convolutional encoder-decoder or convolutional autoencoder (CAE) network. In this case a filter of $\theta_h = 0.01\ m$ was used. To investigate the influence of the skip connections that have been introduced to the modified U-Net architecture, we compare the results on the test simulation to the convolutional encoder-decoder network that does not have skip connections. It can be seen that the modified U-Net architecture has smaller RMSE in h and $\mathbf{u}$ than the convolutional encoder-decoder network. Also, the flood extent in time is more close to the SOBEK simulation. This indicates that skip connections are effective for training the deep learning architectures and for emulating flood simulation. From the plot indicating the RMSE in h_{wet}, we see that the modified U-Net is first more accurate than the convolutional encoder-decoder network however at around 10 time steps it starts the RMSE becomes larger. The modified U-Net may preserve more detail of the flood than the convolutional encoder-decoder network but these results indicate that this also result in a larger error propagation.

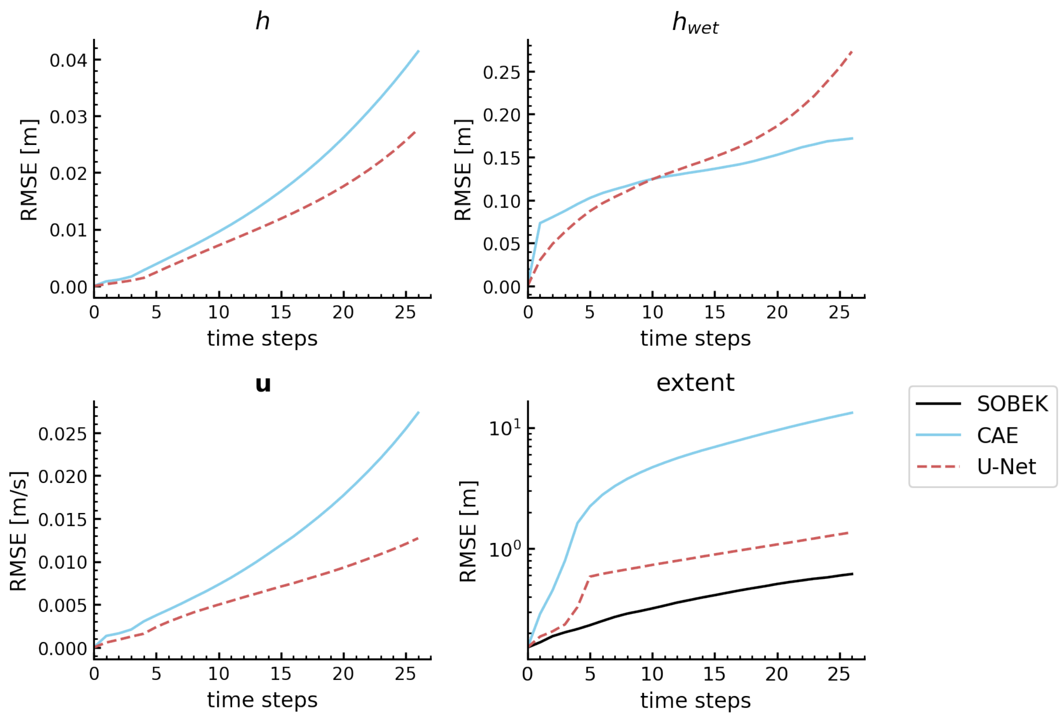


Figure 6.5: **Test simulation performance of the CAE and U-Net architectures**.

6.2.2. Influence of time step

By looking at the patch-level examples with time shift $\Delta t = 1\ min$, it was found that the change in flow characteristics $\Delta\phi = \phi_{t+1} - \phi_t$ between two consecutive time steps was often very small. In training, the network architectures may therefore focus mainly on reconstructing the input and not on the gentle difference in flood propagation. Increasing the time steps between frames in the examples may make it easier for the networks to learn these temporal changes. Here, we test the performance of the U-Net network if we increase the time step of training examples from 1 minute to 2 minutes. Figure 6.6 presents the performance of the modified U-Net architecture trained with time step $\Delta t = 1\ min$ and time step $\Delta t = 2\ min$. It can be seen that the network trained with 1 minute time steps is has smaller RMSE for h and $\mathbf{u}$ and also follows the flood extent of the SOBEK simulation much more closely than the network trained on time steps that are twice as large. From the plot showing the RMSE in h_{wet}, wet region of the SOBEK simulation, it can be seen that the network trained on two minute time steps performs better. For the same duration, the number of predictions created with the surrogate is however twice as small compared to the network trained on 1 minute time steps and thus it is effected less by error propagation. The plots indicate that increasing the time step of training examples does not lead to more accurate surrogate simulations. Therefore, a time step of $\Delta t = 1\ min$ will be used in the surrogate simulations in the remaining of this study.

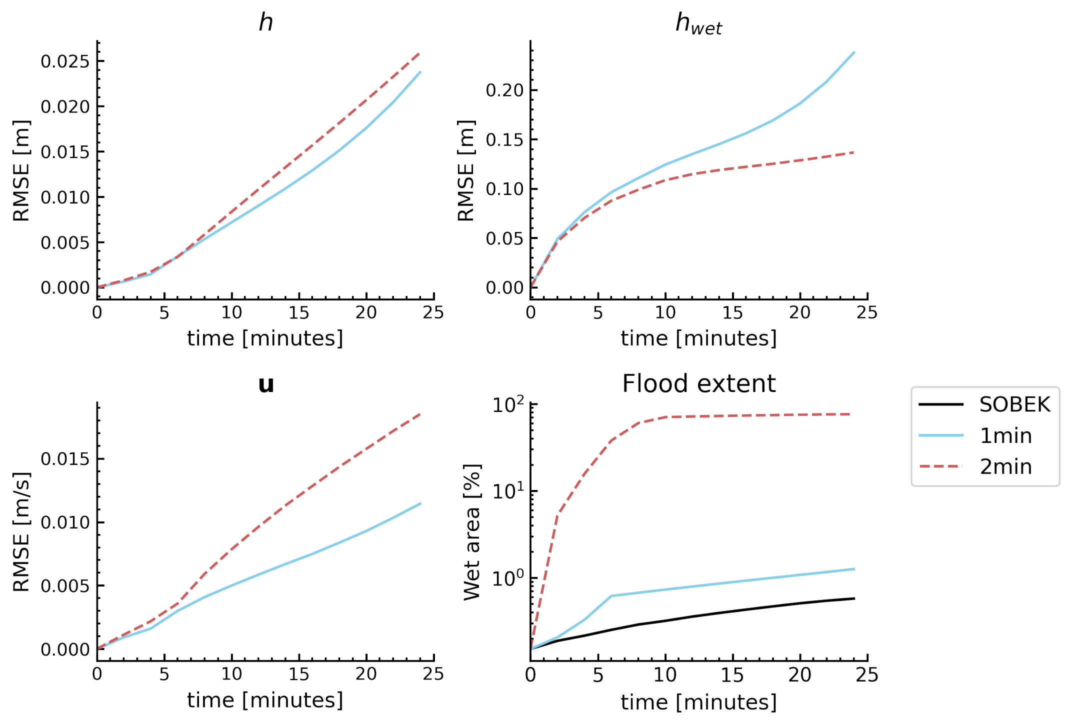


Figure 6.6: **Test simulation error for U-Net architecture trained on different time resolutions**.

6.2.3. Physics-informed loss

In this section a loss functions is proposed that aims to use knowledge about the physics of floods in the training process. The loss function consists of two terms. The first term is a weighted mean squared error loss that distinguishes between wet and dry parts of the area. From the results of the training runs using a MSE loss term, the trained networks were found to often falsely predict water depths in regions that are dry in the SOBEK flood simulations. Here we use the snapshots of SOBEK at time $t+1$ to create a wet-dry mask. This mask is then used to compute the MSE in the dry regions and the MSE in the wet regions. The weight λ_w and λ_d for are then multiplied with the wet MSE and dry MSE respectively.

$$L_1 = \lambda_w MSE(Y,\hat{Y})_{wet} + \lambda_d MSE(Y,\hat{Y})_{dry} \tag{6.1}$$

If we choose the weight λ_d larger than λ_w, the network is forced to focus more on correctly predicting dry regions in order to minimize the loss function. Figure 6.7 shows a wet-dry mask for a certain example in the training set.

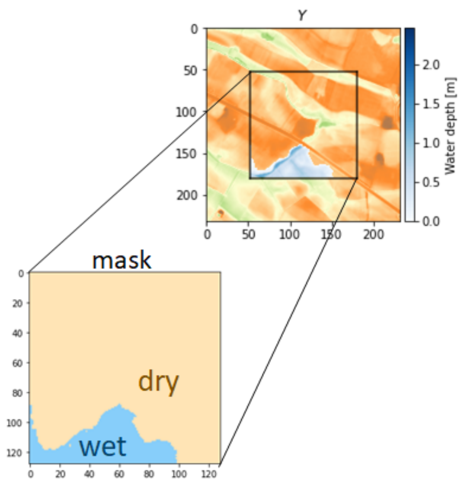


Figure 6.7: **Mask used in the wet-dry loss function**. The ground truth Y at a certain time step t can be used to extract a mask of wet- and dry regions.

The second loss term that is proposed here is based on the continuity equation, the first equation of the shallow water equations.

$$\frac{\partial \zeta}{\partial t} + \frac{\partial hu}{\partial x} + \frac{\partial hv}{\partial y} \tag{6.2}$$

This is a mass balance, which for incompressible fluids is equal to the volume balance. Violation of this equation means that the volume of water between two time steps is not conserved and thus water has appeared or disappeared in one time step. Since this is physically not possible, we can use this violation as a measure to penalized the network.

In training, the input $X_t = (h_t, u_t, v_t)$ can be used to create a prediction with the network $\hat{Y}_{t+1} = (\hat{h}_{t+1}, \hat{u}_{t+1}, \hat{v}_{t+1})$. The time derivative in eq. 6.2 can then be approximated with simple backward difference. The spatial derivatives can be computed using a convolutional layer with a specific kernel. A partial derivative $\frac{\partial f}{\partial x}$ can be computed with the kernel weights $[-1, 1]$ as a row vector and $\frac{\partial f}{\partial y}$ with the same weights as a column vector. For the spatial derivatives, the product rule of differentiation can be used.

$$L_2 = \lambda_c \frac{1}{n} \sum_{\Omega} \left(\frac{h_{t+1} - h_t}{\Delta t} + h_t \frac{u_t}{\Delta x} + u_t \frac{h_t}{\Delta x} + h_t \frac{v_t}{\Delta y} + v_t \frac{h_t}{\Delta y} \right)^2 \tag{6.3}$$

Figure 6.8 present a visualization of the first term in this loss function. In this case, the water depth h_{t+1} of a SOBEK simulation is shown. During training, the network is used instead to compute the prediction $\hat{h}_{t+1}$.

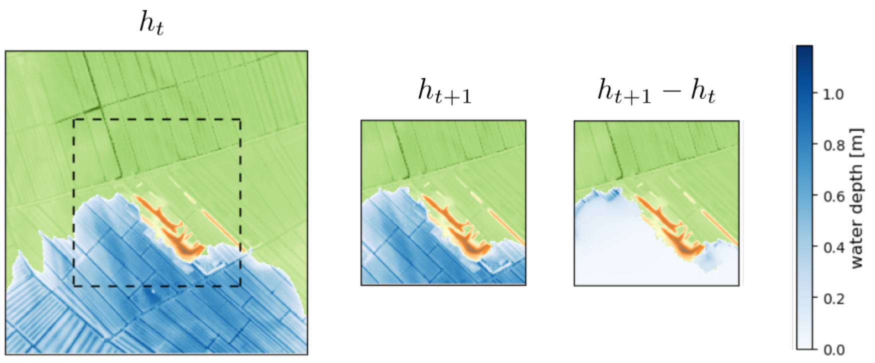


Figure 6.8: **Visualization of the water depth difference** $ht+1 - h_t$ **between two time steps**. Here, the water depth h_t over an input patch of dimension 232 × 232 is presented and the water depth h_{t+1} over the output patch with dimension 128 × 128 pixels.

We trained the modified U-net architecture to predict $(h_{t+\Delta t}, u_{t+\Delta t}, v_{t+\Delta t})$ with a 1 minute time step and the loss $L = L_1 + L2$. For the weights, $\lambda_w = 0.2$, $\lambda_d = 0.7$ and $\lambda_c = 0.1$ were chosen. It was found that a small value for λ_c resulted in better training results. From Figure 6.9, it can be seen that the use of this physics-informed loss function for training results in a larger RMSE in h and $\mathbf{u}$. The resulting surrogate simulation is also less accurate in terms of the flood extent compared to the training run where a MSE loss term was used. The use of a weighted loss function may complicate the training process and it may be more difficult to find a minimum for the loss function. Also, SOBEK uses a smaller time step internally than the time step of 1 minute for which output is available. Therefore, the continuity loss term may sometimes not be very accurate in some cases.

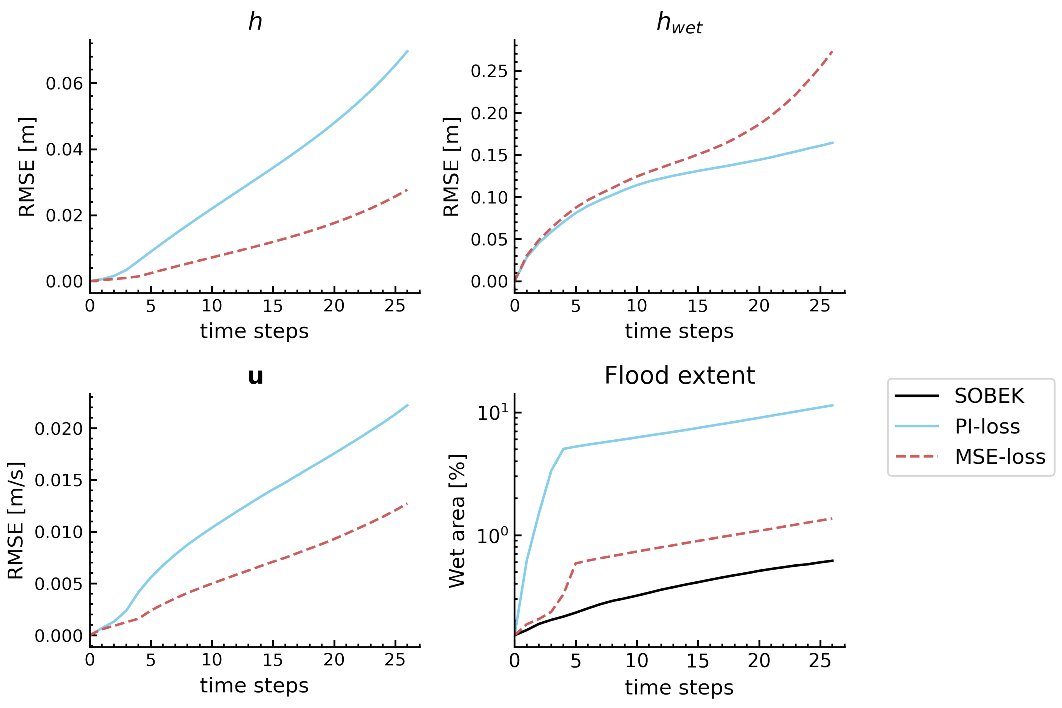


Figure 6.9: **Test simulation error for U-Net architecture trained with MSE loss and PI-loss**.

6.3. Patch predictions

To see the output of the surrogate simulations created by the trained CAE and U-Net architectures and compare them with the SOBEK simulation, we can take a closer at one of the patches of the test simulation. Figure 6.10 shows one of the patches in the test simulation at time $t = 50$ minutes. The input patch is indicated with a solid black line and the corresponding output patch with a dashed line.

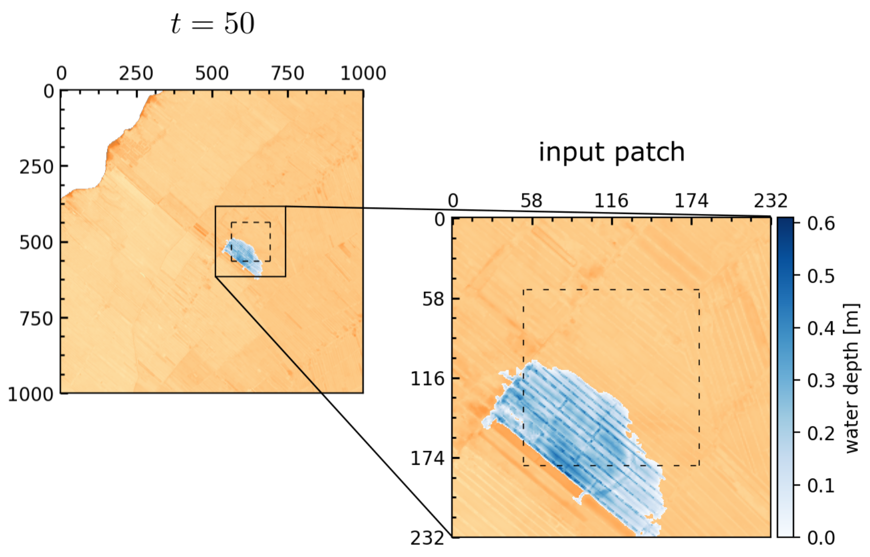


Figure 6.10: **Patch 25 in test simulation**. Solid line shows input patch and the dashed line shows the output patch at time $t = 50$ minutes.

Figure 6.11 presents three patch-level frames of the surrogate flood simulations and SOBEK simulation. Here, the trained CAE network and modified U-Net architecture were used to create the surrogate simulations. The results are presented for three forecast times: 4, 8 and 14 time steps and starting at $t_0 = 50\ min$. It can be seen that the predictions from the CAE network are distorted and blurry as expected due to loss of information resulting from the pooling and upsampling layers. At $t_0 + 12\ min$, there is some propagation of the flood however also dry regions appear that were wet originally. For the modified U-Net architecture, the information loss is reduced by using skip connections that copy information from the encoder to the decoder part in the network. It can be seen that the output of this network preserves a lot more detail. It can be seen however that there is less flood propagation and the water depths increase faster.

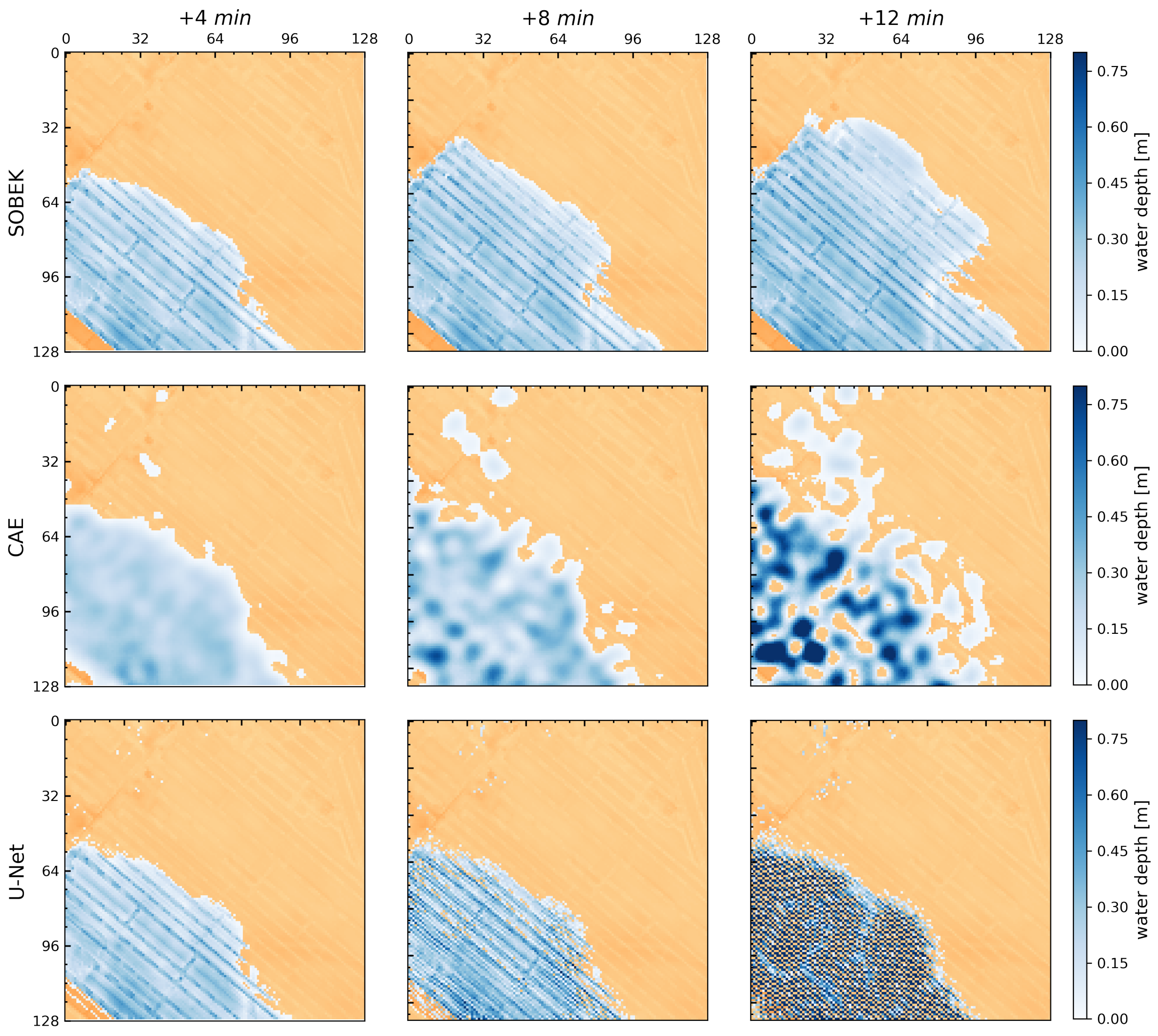


Figure 6.11: **Water depth predictions for a flow patch in the test simulation**. Here a begin time $t_0 = 50$ min is used and the 2 min, 14 min and 28 min forecast is presented.

6.4. Predicting flow discrepancies

In this section we discuss performance of the convolutional encoder-decoder and modified U-Net architectures trained on flow discrepancy examples. We discussed that flow characteristics between frames in a simulation (with 1 minute time step) is found to be small. In this section we investigate if predicting flow discrepancies $\Delta\phi$ between two time steps leads to more efficient training and better performance. Future frames of the surrogate simulation ϕ_{t+1} can be created by taking the current frames ϕ_t and adding the discrepancies predicted by the trained network $\Delta\phi$.

$$\phi_{t+1} = \phi_t + \Delta\phi \tag{6.4}$$

By looking at the resulting surrogate simulations we found that without a threshold filter, small errors propagate to neighbouring patches quickly and spread over the complete computational domain within a few time steps. Therefore, we investigate the use of an error filter, but now on the predicted discrepancies $\Delta\hat{h}_t$. Figure 7.2 presents the error in the surrogate simulation created with the trained convolutional encoder-decoder architecture for different threshold values θ.

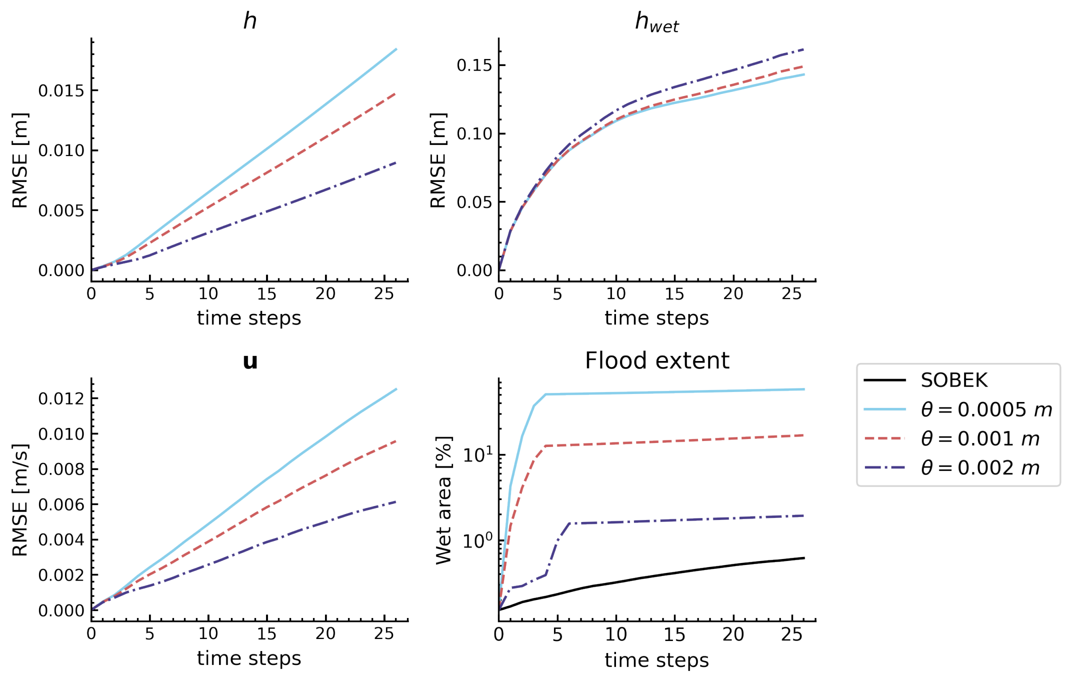


Figure 6.12: **Test simulation error for CAE network trained on flow discrepancy $\Delta\phi$ prediction and influence of noise filter with different water depth threshold θ.**

It can be seen that a threshold value of $\theta = 0.002\ m$ gives the smallest RMSE in h and $\mathbf{u}$ and it also is closest to SOBEK in terms of flood extent in time. The RMSE over the wet cells in the SOBEK simulation h_{wet} is however slightly larger than for smaller threshold values.

6.5. Comparing feed-forward architectures

In this section, we compare the performance of the modified U-Net and convolutional encoder-decoder network trained on predicting ϕ_{t+1} and $\Delta\phi$. We use the error filters as discussed in previous sections, a time step of 1 minute and the networks were trained with a MSE loss function. Figure 6.13 presents the performance of these trained architectures. It can be seen that the surrogates that predict the flow discrepancies give simulations that have the smallest error in time. This indicates that it is more efficient to train the networks on the flow discrepancy examples. In terms of flood extent, we see that the convolutional encoder-decoder network or CAE performs better in the case of flow discrepancy predictions. The opposite is true for the modified U-Net architecture. If we compare all four surrogates, the CAE trained on flow discrepancies gives the best overall performance.

In Figure 6.14, the frames of the surrogate simulations at different time steps after $t_0 = 50\ min$ are presented as well as the ground truth SOBEK simulation. Here the water depth predictions are presented for the CAE and U-Net architectures trained on predicting the flow discrepancies $\Delta\phi$. It can be seen that the predictions of the U-Net and CAE architectures have difficulty predicting the propagation of the flood, especially in the upper right direction. The modified U-Net architecture seems to predict slightly more small errors than the convolutional encoder-decoder network.

The various trained network architectures were found to have similar calculation time. For a flood simulation with 316 time steps, the surrogate takes less than one minute to finish on a CPU. On a GPU the run time of the surrogate simulation is even and the efficiency of the surrogate model may be still improved to reduce the run time even further. The calculation time of the test simulation created with SOBEK was found to be a little more than 1 hour for the same number of frames.

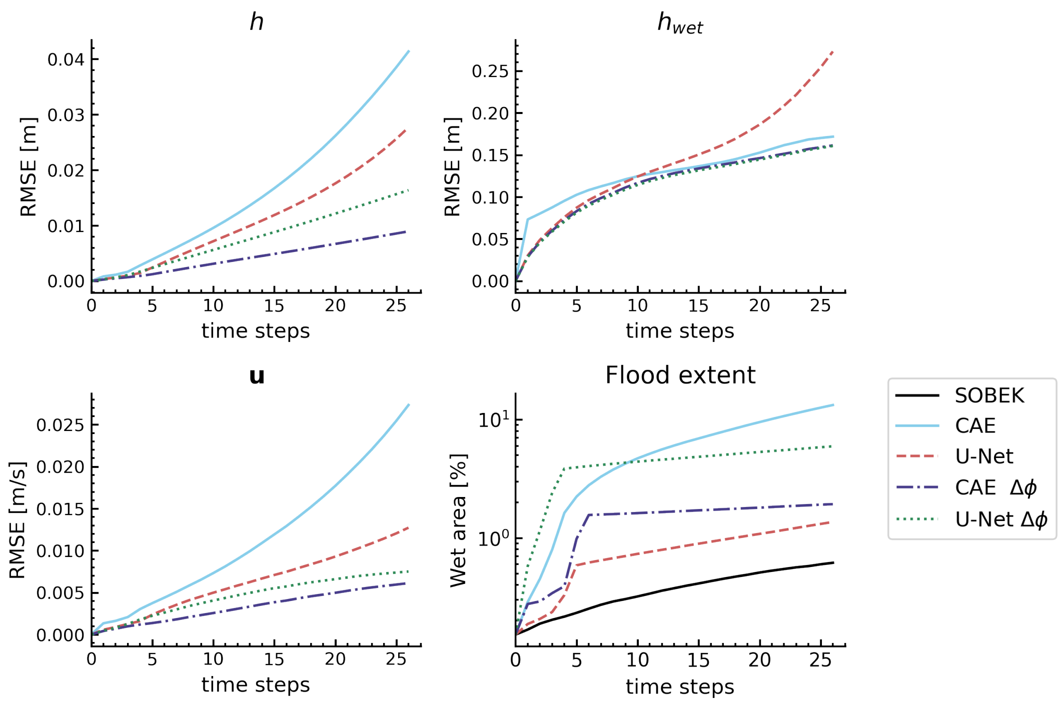


Figure 6.13: **Test simulation performance of trained architectures**. Here the CAE and U-Net are the ones trained on 1 min timesteps with MSE loss. CAE $\Delta\phi$ and U-net $\Delta\phi$ indicate networks that predict flow discrepancies.

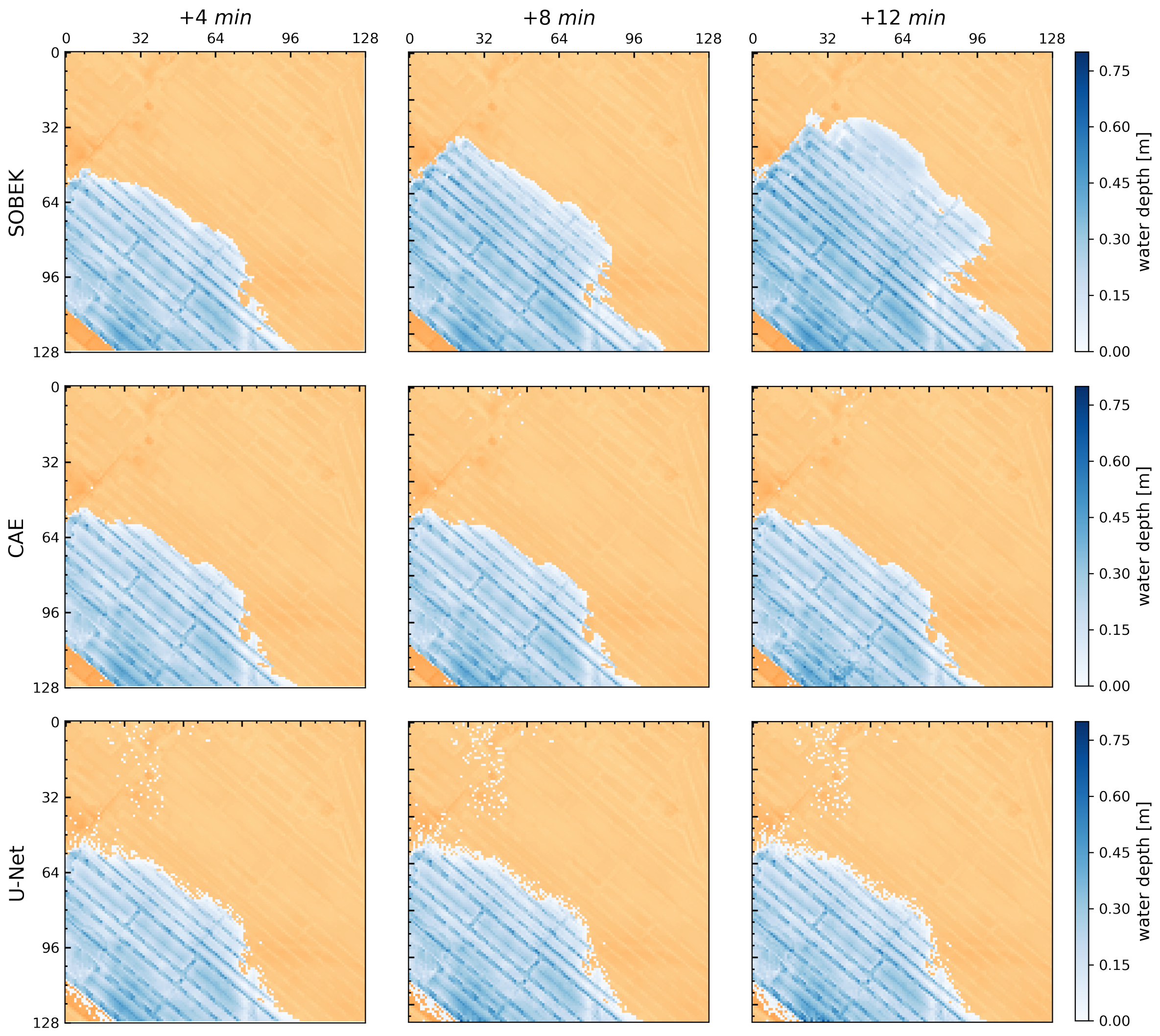


Figure 6.14: **Water depth predictions for a flow patch in the test simulation**. Here a begin time $t_0 = 50\ min$ is used and the 4, 8 and 12 minute forecast is presented.

7
Recurrent architecture

In this Chapter the performance of the surrogate model is discussed using a trained convolutional Long-Short Term Memory (ConvLSTM) architecture. This is a recurrent network architecture that takes as input a sequence of past flow characteristics $X_{flow} = [\phi_{t-3}, \phi_{t-2}, \phi_{t-1}, \phi_t]$ instead of a single frame. This input sequence is processed by a flow encoder. A separate terrain encoder is used for the elevation and roughness map of an input patch. The network layout is presented in Figure 4.9. Here, performance of the network on the test simulation is discussed. After that, a small case study is presented in which the surrogate model is used to create simulations for a dike ring in the Netherlands. These surrogate simulations are compared to flood simulations created with SOBEK. We show that the surrogate model is able to create surrogate flood simulations for dike ring areas. Finally, we discuss the calculation time of the surrogate model and compare it to the calculation time of the SOBEK flood simulations.

7.1. test set performance

The ConvLSTM network consists two separate encoder networks. This network architecture takes as input a sequence of past flow predictions. The sequence length of 4 used in this study, which was found be possible with the size of GPU memory used in training. The terrain of a patches does not change in time. This means that the elevation and roughness are the same for all flow frames in the sequence. Therefore, it is decided split the flow and terrain inputs and use separate encoder networks to avoid redundancy in the inputs. Figure 7.1 presents the Root Mean Squared Error (RMSE) in the water depth h and flow velocity $\mathbf{u} = \sqrt{u^2 + v^2}$ for the surrogate simulation for different starting times $t_0 \in [0, 10, 50, 100]$ minutes. The RMSE in h remains within 10 cm for about 22 time steps and does not show a dependence on the begin time t_0 of the simulation. It stays very close to zero for the first 15 time steps, indicating that the predictions for this forecast duration are accurate. In the flow velocity $\mathbf{u}$, it can be seen that the RMSE increases more rapidly for the larger begin time of $t_0 = 100$ minutes.

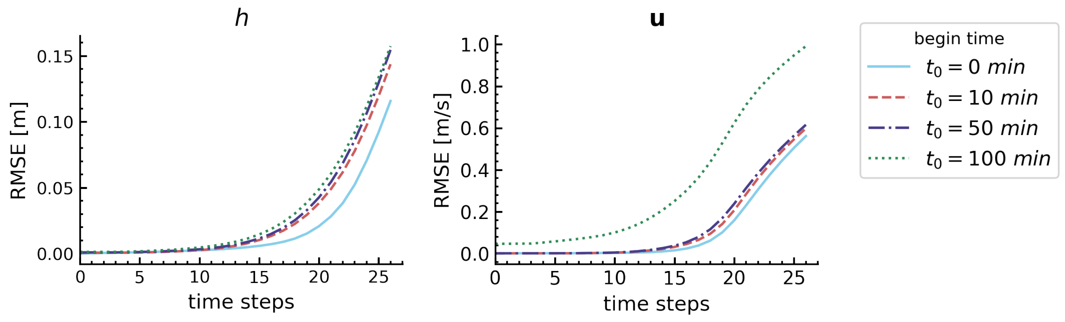


Figure 7.1: **Test simulation RMSE for ConvLSTM network**. The error in water depth h and flow velocity $\mathbf{u} = \sqrt{u^2 + v^2}$ is presented for different begin times t_0.

From observation of the predictions, we can see that small errors are again present in the predictions of the network at locations that are not connected to the flood extent. To reduce these errors we again apply an error filter for various filter threshold values θ_h. Here, it can be seen that using a filter of $\theta_h = 0.002\ m$ is very effective in reducing the RMSE in the water depth h and in $\mathbf{u}$. In addition, the flood extent is very close to the SOBEK simulation which is considered as the ground truth. If we look at the error in h_{wet}, that is in the wet cells of the SOBEK simulations, it can be seen that this larger threshold value leads to larger a larger RMSE. However, the errors in time of the various simulations are close. These results indicate that using a filter of $\theta_h = 0.002\ m$ is most effective and for that reason we will use this value in the application of the surrogate model.

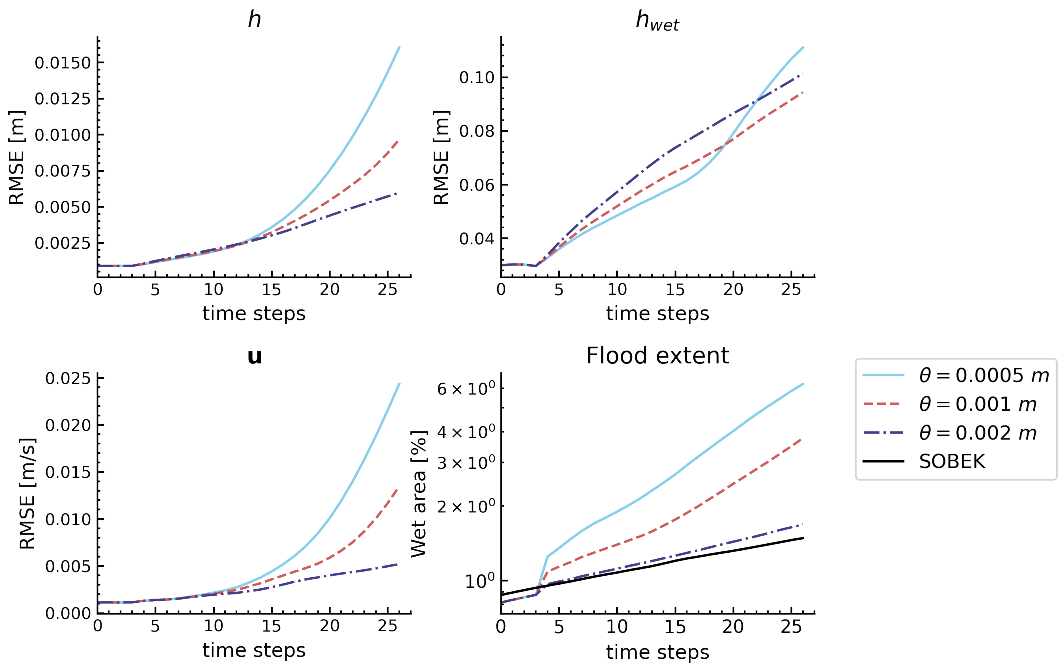


Figure 7.2: **Test simulation error for ConvLSTM network trained on flow discrepancy $\Delta\phi$ prediction and influence of noise filter with different water depth threshold θ.**

7.2. Case study: flood simulation in Arcen

In this section, we discuss a case study where we applied the surrogate to make flood simulations in a small levee system located along the river Meuse near Arcen in Limburg, the Netherlands. The results of the surrogate model are compared to the hydrodynamic flood simulations created with SOBEK, which are considered to be ground truth. The main purpose of this case study is to show that the surrogate model and the method of patch-level predictions can be used to create flood simulations for dike ring areas of arbitrary shape. The terrain of the dike ring area has a lot of height difference, making it different from other terrain models used in the training set. For that reason makes a suitable case for testing the generalization capabilities of the surrogate. The relatively small size of the levee system allows for the computation of the flood using the hydrodynamic model within reasonable run time and memory use. In total, three simulations will be created in Levee system 65 in the South-East of the Netherlands. Figure 7.5 presents the study area with the breach locations along the river as well as the grid with input and output patches used by the surrogate model.

The levee system lies along the upper part of the Meuse such that it is not influenced by the tide or backwater effects from lakes. High water levels resulting from discharges in this part of the river are due to heavy rainfall events. The river is slightly meandering in this region of the Netherlands. There is some urban area in the middle of the levee system and further here is mostly agricultural businesses in the region and recreational destinations. To the North, East and South of the area, high elevation grounds are present and no flood protection is needed. The accepted frequency of high water in this region of the Netherlands used to be 1/50 years, however recently the protection level was increased to 1/250 resulting in a dike strengthening project called Maaswerken.

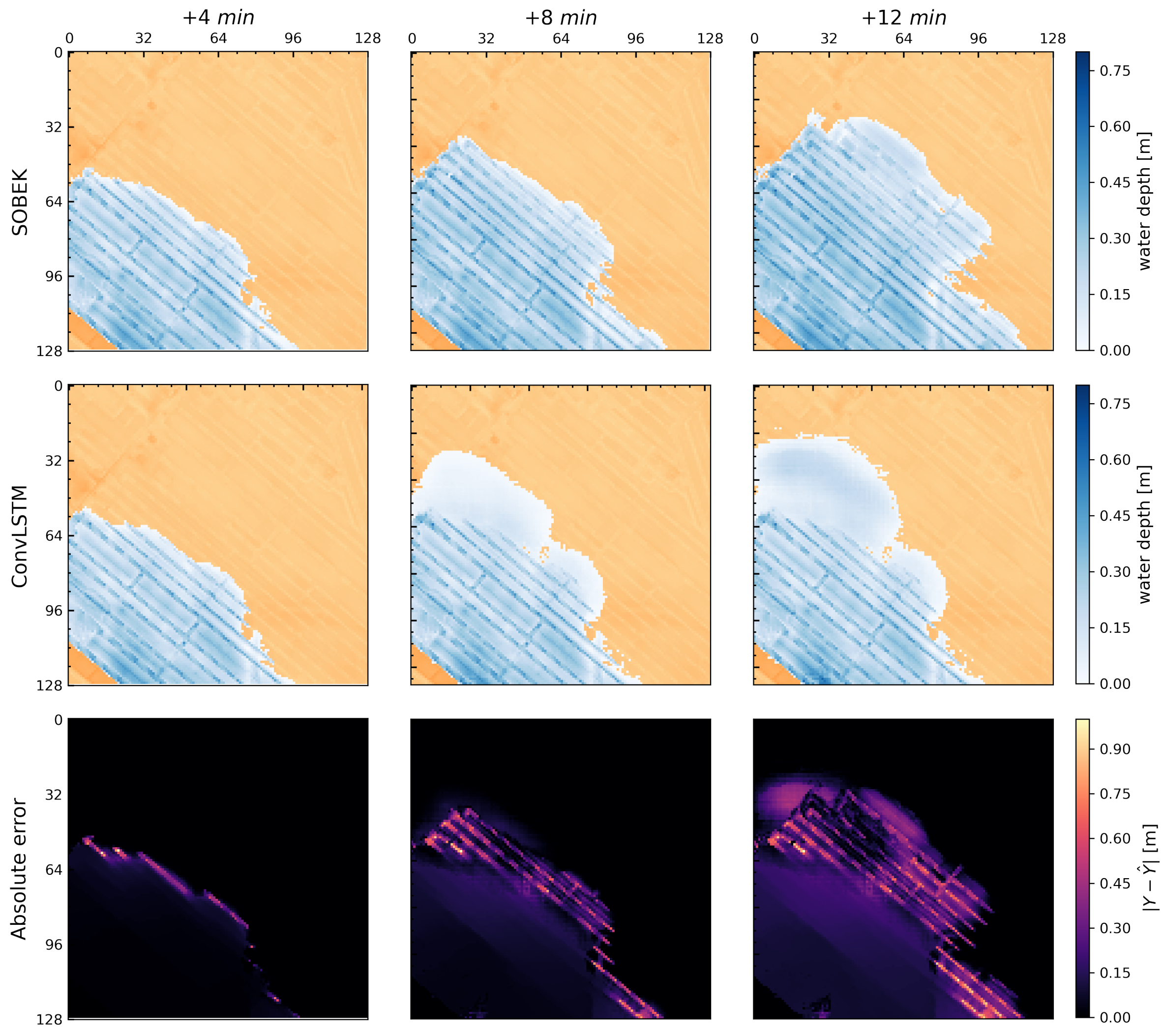


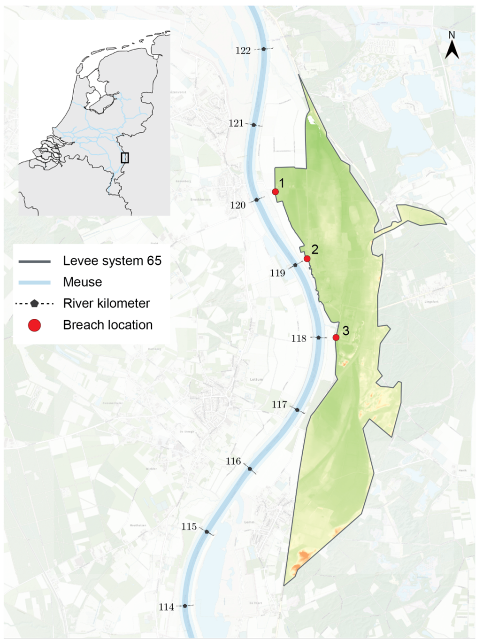
Figure 7.3: **Water depth predictions for a flow patch in the test simulation**. Here a begin time $t_0 = 50\ min$ is used and the 2 min, 14 min and 28 min forecast is presented.

Table 7.1: **Scenario specifications**. Breach location and boundary conditions for three flood scenario's in Arcen.

variable	UNIT	**breach 1**	**breach 2**	**breach 3**
Kmr	km	120	119	118
Latitude	DEG N	388787.72	387932.65	387058.04
Longitude	DEG E	209477.48	209893.02	210282.55
Breach cell elevation	m + NAP	15.72	16.51	16.35
Peak (1/300 Y)	m + NAP	17.33	17.41	17.54

7.2.1. Hydraulic boundary conditions

To test application of the surrogate model we use realistic boundary conditions. Here three breach locations that are chosen based with water levels based the safety norms in the Netherlands. The safety norm along the dikes in the Arcen levee system have return period 1/300 years. Hydraulic boundary conditions for locations along the Meuse were quantified in a study performed by Deltares [5]. A 1D hydrodynamic model was used with an upstream boundary point at Borgharen. For this upstream boundary condition, generated time series of the GRADE project were used instead of extrapolation from historic high river discharge observations. The advantages of using generated series is that they avoids the need to choose an extreme value distribution



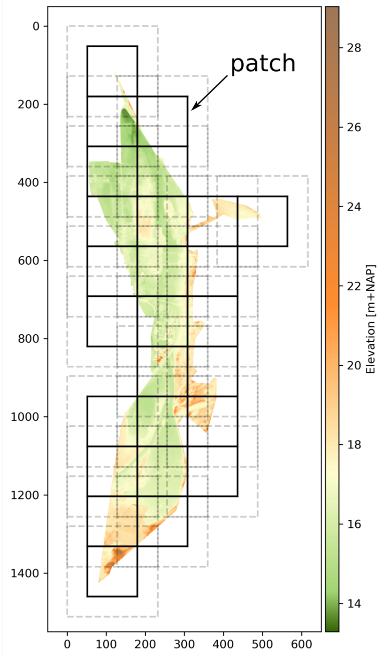


Figure 7.4: **Topographic map of levee system 65 near Arcen and grid for the surrogate model**. Three breach locations are indicated that will be assessed in this case study. The river kilometers indicate the distance from the Borgharen measuring station. The input and output patches are shown in a grid that is used by the surrogate model. Here, output patches are presented with black solid lines and input patches with dashed line.

and fit in on the arguably too small and possibly inhomogeneous set of historic distribution. GRADE uses a stochastic weather generator using nearest-neighbour resampling and a hydrological rainfall-runoff model as well as a routing model (SOBEK 1D) for the river Meuse basin to the simulate a 50,000 year discharge series at Borgharen. All water level series that can occur downstream of this location are assumed to be due to the change of discharge in Borgharen as a function of time. In this stretch of the river, the water level occurring is primarily due to high river discharges and no tidal or backwater effects are present.

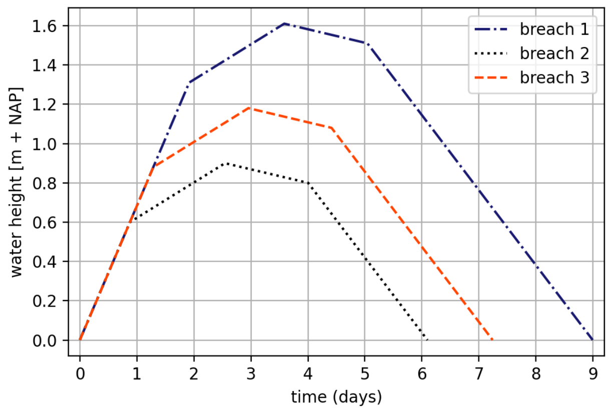


Figure 7.5: **Hydrograph boundary conditions specified at the three breach locations**. These hydrographs were constructed using water levels corresponding to a 1/300 year return period as determined in [5].

7.2.2. Surrogate model setup

In this section, we discuss several steps that are required to use the surrogate model. We first discuss how to construct a grid of input and output patches. Second, we discuss how to specify the boundary condition in the breach location grid cell. Finally, some pre-processing steps are explained that are needed to use the surrogate model for the dike ring area.

Define a computation grid

The first step in creating the surrogate flood simulation for the Arcen dike ring is to create a static grid of patches. The choice for the location op patches within the grid is made based on the breach location. Here we choose the breach location to lie approximately in the center of a patch, and build patches around it outwards. The result of such creation of a grid is illustrated in Figure 7.5. Note that some patches are contain only a very small region of the levee system. This is due to the chosen patch size and due to the shape of the Arcen levee system. It may be a challenge for application of the surrogate model, because in the proposed training procedure, flow examples in such patches were not present.

Specify initial- and boundary conditions

To specify specify the hydrograph boundary condition, the water depth variation is stored as a one-dimensional array with a length determined by the number of time steps in the simulation. The information of this boundary condition is given to the model by adjusting the water depth in the cell at the breach location. This means that the patch containing the breach location is informed on water level differences, and all other patches get this information indirectly by flow through the patch boundaries. Another important step is to adjust the elevation map of the area such that the elevation at the breach location is the datum or zero-mark. This same pre-processing step was done during training, and it avoids having very different ranges of inputs. In addition, no data values which sometimes are chosen -999 in DEMs should be adjusted to zeros, again to match training conditions.

7.2.3. Results

In this section, the results of flood simulations are presented for the proposed case study. We first discuss the SOBEK flood simulations that were made for the three breach scenario's. Second, we compare the surrogate flood simulations to the SOBEK simulations to evaluate the performance of the surrogate model.

SOBEK simulations

These results were that were created with SOBEK using a 2D breach node with the above described hydrographs. The simulation output are water depths and flow velocities at 1 minute time intervals. These are returned in ascii file format. The total number of ascii files that can be returned is 10,000 so that the simulation was stopped after that amount of minutes. It should be noted that the size of this data is quit large (hundreds of GBs). Also generating these simulations was found to be very computationally intensive due to the small mesh width of 5 by 5 meters used in this study. From the different scenario's breach 1 was found to be most computationally intensive. This simulation has not been run for the full prescribed duration as the computation took very long, it was stopped after running for over one month.

Figure 7.6 presents the maximum water depth maps for the previously defined flood scenario's. It can be seen that the low lying part in the North of the flood prone area has the largest water depths. These water depths can be over 3.5 meters, for the first flood scenario.

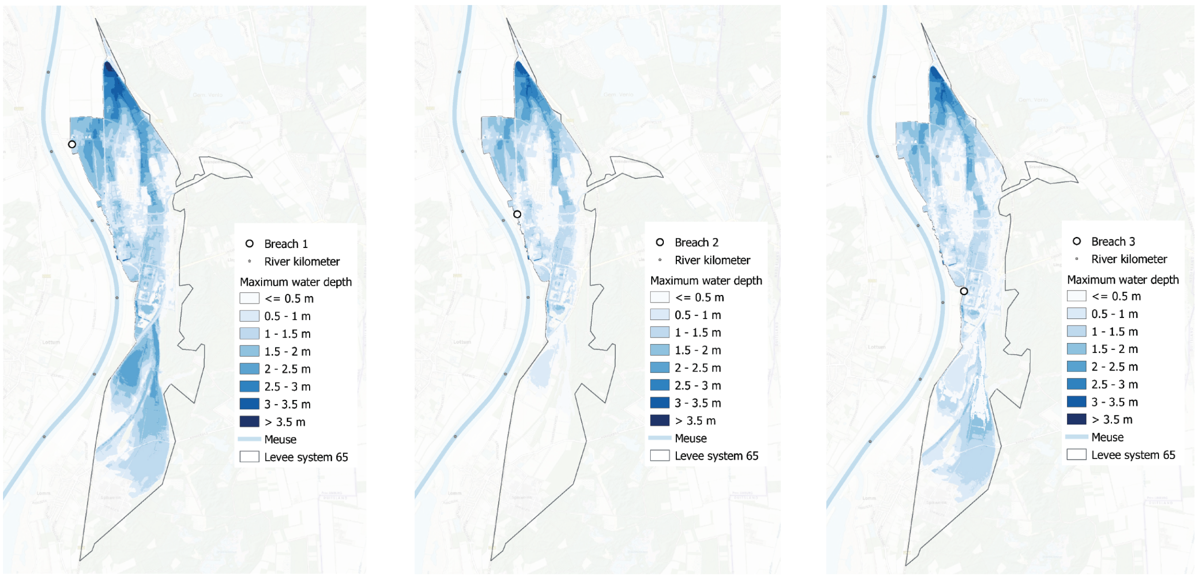


Figure 7.6: **Results of maximum water depths in the Arcen area for the three flood scenarios**.

Figure 7.7 presents maps with the maximum occurring flow velocities in the three flood scenario's. It can be seen that in the first scenario the largest flow velocities are encountered. This explains the long run time of this simulation. SOBEK internally adjusts the time step of computation based on the Currant number to ensure numerically stable solutions. When the flow velocities are high, a smaller time step is chosen resulting in longer run time. The larger flow velocities were expected as the amount of water entering the flood prone area was largest for this scenario. In the second and third scenario, the maximum flow velocities are much lower.

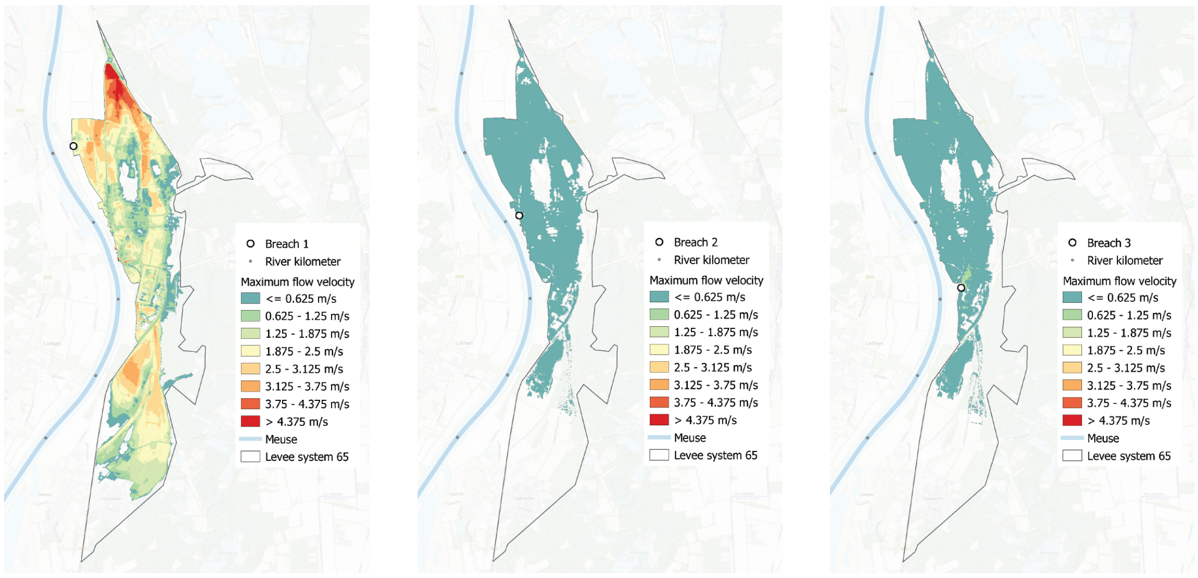


Figure 7.7: **Results of maximum flow velocities in the Arcen area for the three flood scenarios**.

Surrogate simulations

We use the trained ConvLSTM network to create three surrogate flood simulations and compare these to the three simulations made with SOBEK. Figure 7.9 presents the RMSE of the surrogate model for the three flood scenario's as a function of the forecast time steps. Here a start time of $t_0 = 100\ min$ was used. It can be seen that the surrogate model performs better for the second and third simulations and it has difficulty with accurately emulating the first simulations. The flow velocities in the latter flood simulation were found to be much larger and this may be more difficult for the surrogate model because the flood propagates faster.

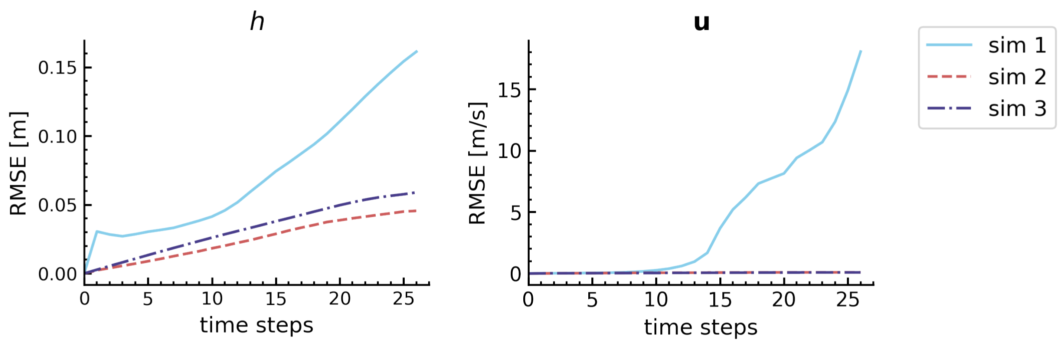


Figure 7.8: **RMSE for three surrogate simulations**. The error in water depth h and flow velocity $\mathbf{u} = \sqrt{u^2 + v^2}$ is presented for begin times $t_0 = 100\ min$.

Figure 7.9 shows the flood extent the flood simulation made with SOBEK and the surrogate flood simulation created with the trained surrogate model. Again, the flood in the first simulation spreads much more rapidly and the surrogate model is not able to replicate this behaviour accurately. The performance of the surrogate model for the second simulation are much better. For this scenario, the flood extent in time of the surrogate simulation is much closer to the SOBEK simulation. Finally, in the third simulation we see that the flood simulation resulting from the surrogate model first spreads to fast over the domain. However, after approximately three time steps the increase in flood extent is very similar to that of the SOBEK simulation.

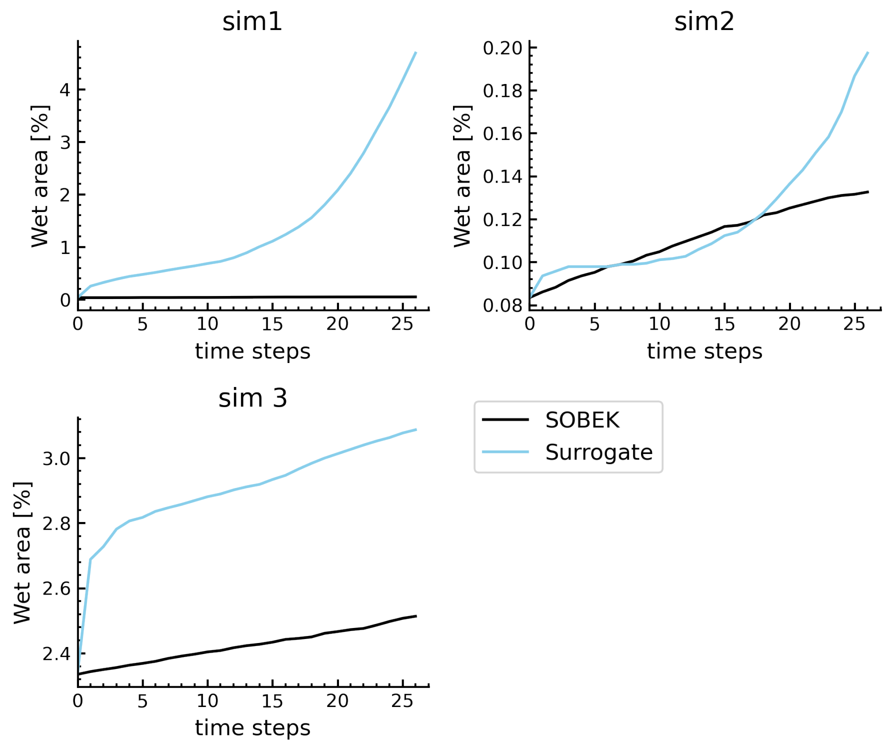


Figure 7.9: **Flood extent in time for surrogate and SBOEK simulations**.

In Figure 7.10, we present the results of surrogate model for the output patch surrounding the breach location. We show the predictions of the surrogate model for the water depth after 8, 16 and 24 forecasting time steps.

7.3. Calculation time

In this section we discuss the simulation run time of the surrogate model and compare it to the simulation run time of the SOBEK simulation. The different SOBEK simulations were found to have a lot of variability in calculation time. The duration of these simulations was over 12000 time steps, however it was found that SOBEK has a limit on the amount of simulation snapshot it can output of 9999. The first was found to have a very long run time. This simulation was stopped after running for 2 months. This long calculation time may be explained by the large flow velocities that were present in the simulations. SOBEK adjust the time steps used based on a stability condition, for large flow velocities a small time step is required for numerically stable solutions. The second and third simulations were found to have a run time of 3 and 5 days respectively.

The recurrent network of the surrogate model takes longer to run than the feed-forward network types. The reason for this is that the architecture is more complex than a feed-forward network, it has more weights. In addition, this network takes a sequence of past flow snapshots as inputs and thus it takes longer to load these arrays into memory. The run time of these simulations was found to be still smaller than the SOBEK simulations. It was not possible to create a flood simulation of 9999 time steps since that would require over 200 Gigabytes of CPU memory. The calculation time of 999 time steps was found to be approximately 2 minutes on a CPU. When using a GPU for making the surrogate flood simulation, the calculation time is expected to be even much lower. Also, it is expected that there is still room for improving the efficiency of the surrogate model.

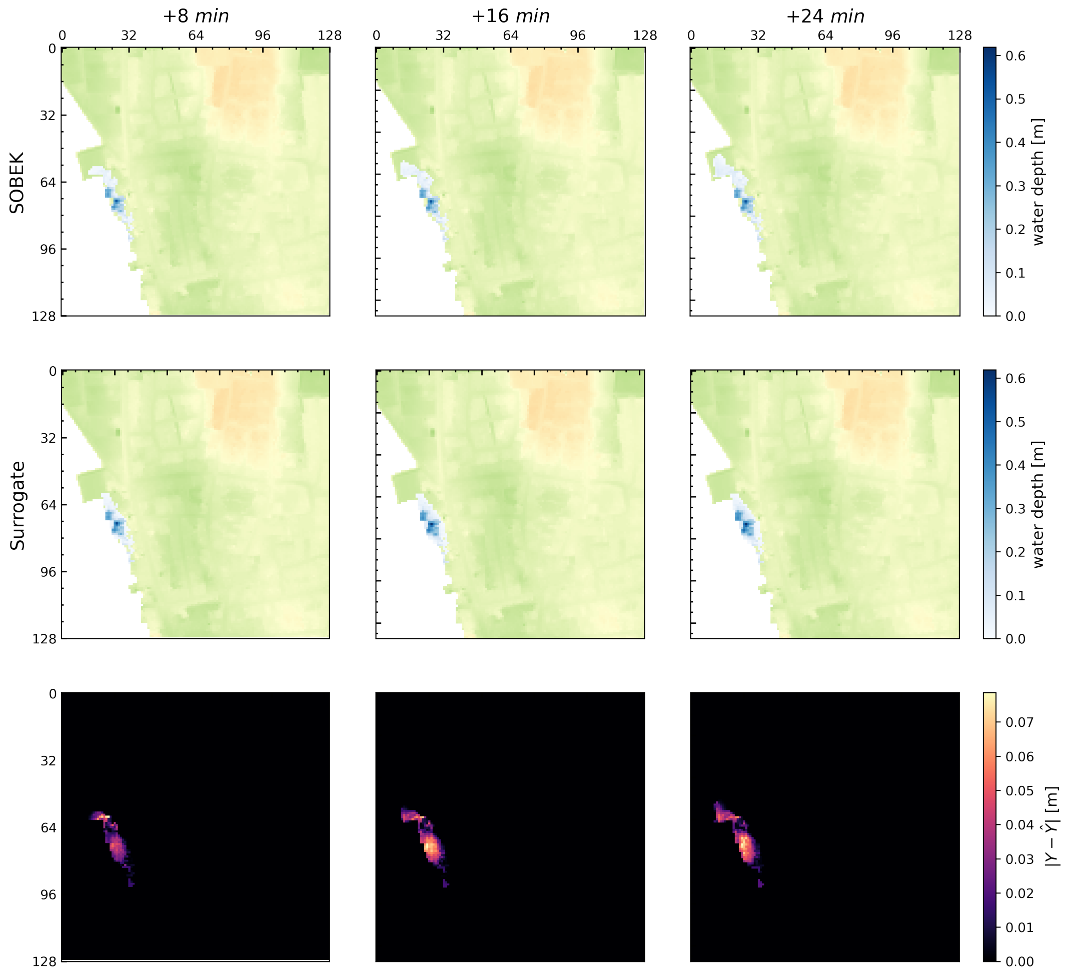


Figure 7.10: **Water depth predictions for a flow patch in sim 2**. Here a begin time $t_0 = 5000\ min$ is used and the 8 min, 16 min and 24 min forecast is presented.

8 Discussion

The proposed framework for surrogate modelling of flood simulation can be used to train deep neural networks to emulate such simulations. A trained surrogate model may be used as a substitute of expensive hydrodynamic flood simulations and it is orders of magnitude faster in terms of calculation time. This has the potential to be used for real-time flood forecasting allowing for example scenario-specific evacuation planning. In addition, such surrogates may be used in a Monte Carlo framework to quantify uncertainty in consequences of flood by simulating a very large number of possible flood scenario's.

In this study, convolutional neural networks with an encoder-decoder structure were trained on examples consisting of high resolution flood simulation output. Some assumptions and choices were made in constructing the data set and in creating a patch-wise training environment that is suitable for training the network architectures. These assumptions will be addressed in more detail below.

8.1. Hydrodynamic simulations used in the training data set

In Chapter 4, the input sampling for creating fictitious flood scenario's for the training data set was discussed. For these scenario's, terrain grid tiles from a number of large dike rings in the Netherlands were extracted and used for the training simulations. This makes the trained surrogate models applicable for creating flood simulations in prone areas with similar terrain characteristics. If the terrain of a certain flood prone area is very different from the ones used in training, different terrain may need to be selected for the training simulations to get better performance of the surrogate model.

The mesh width of the uniform grids used in this study was 5 meters which is relatively high. An advantage of using a high resolution grid is that it removes some of the needs to model 1D elements such as channels and water retaining structures inside the 2D domain. From the simulations in the training data set it can for example be seen that the floods sometimes spread very fast through small waterways that have low roughness. If a coarser grid resolution was used, such small waterways would not be present in the terrain model which can lead to a different propagation of the overall flood. The fine mesh width chosen in this study does however require the time step of the simulations to be small for numerical stability reasons. This makes the construction of the training set a time costly process due to long run times of the flood simulations. In addition, the resulting simulation output takes up large storage space. This restricts the number of flood scenario's that can used for the training data set.

The flood simulations were created using the software program SOBEK216 developed by Deltares. The advantage of using this program is that it can be used to write simulation output text files of the flood characteristics every time step (1 minute), which can be used as training data for deep learning architectures. In addition, it uses a rectangular grid which is easy to construct and results in output that has uniform structure which is easy to interpret. The disadvantage of using SOBEK is that it does not have the option to create many simulations automatically, such as it is done with input sampling. The successor of SOBEK, D-Hydro does have such option and may be better suited for creating training simulations. The approach taken in this study is to use a script to adjust certain initialization files with the initial- and boundary conditions that are used by SOBEK to initialize the simulation. However, it was found that this approach was only possible for 2D boundary nodes. For using a breach model or a 1D channel as boundary condition, changing the initialization files was found to be difficult if not impossible. Even though it was possible to change the initialization files for simulations containing a 2D boundary node, it was still required to open the model in the GUI where the model had to be saved manually. This can be a time-costly process and may limit the number of flood scenario's to be defined for the training set. In future study, the successor of SOBEK, that is D-Hydro may be used which is better suitable for defining flood simulations.

In this study, no breach modelling was used and also the outside water level (in the river) was not modelled. The breaches in the training simulations were schematized by triangular hydrographs at a single cell in the square grid domains. The h-t relation at this cell increased to a peak water level after which it decreases to zero after a certain hydrograph duration. The flood simulations would be more realistic if they were combined with a breach growth model and if it was connected to a river model such that the amount of water entering the domain depends on the river discharge. The water level in the river will start to drop as more water enters the flood prone area behind the breach.

From analysing the flood simulations resulting from the set of fictitious flood scenario's created for the training data set, flow velocities in some simulations were found to be very large. The reason for this may be the lack of a river model. The water level in the breach grid cell can increase to 5-6 meter in the most extreme case. Figure 8.1 shows the maximum water depth and maximum flow velocity maps for the simulation with the steepest water level increase and peak of the hydrograph of around 5 meters. It can be seen that the flow velocities in this simulation reach more than 50 meters per second close to the breach location. Near this location, sometimes also instabilities were present

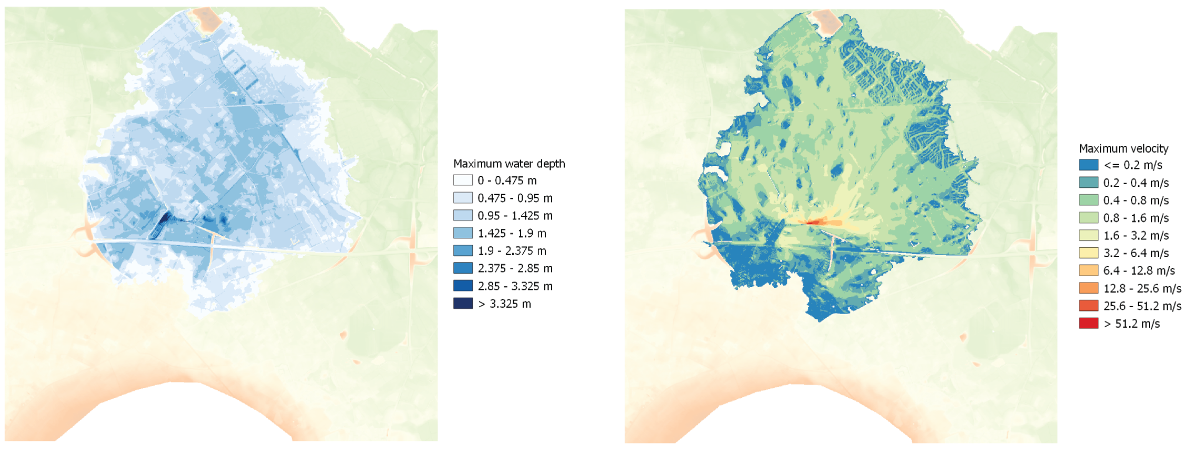


Figure 8.1: **Extreme flow conditions present in training set**. Maximum water depth and maximum flow velocity maps for the training simulation with steepest hydrograph. Large flow velocities are present near the breach location

In some simulations, the water level increase defined by the triangular hydrograph is very gentle which is also not very realistic. In reality the breaches will happen most probably somewhere during the peak discharge or water level. This will result in very rapid flows in the first moments of the flood. Flood simulations in the training set often show to be more gradually varying flows.

8.2. Deep learning architectures and training

In this study, input and output patches were extracted from the expensive 2D simulation output generated with SOBEK. The size of these patches was chosen to be 232 × 232 grid cells for the input patches. This size may be quit large which is beneficial for the efficiency of the surrogate model during application because a smaller number of patches is required to cover the computational domain, for example the dike ring. However, in general it is more difficult to train models using high-dimensional input. The performance of the model may increase when smaller patches are used. In addition, the complexity of the simulations was also increased because a grid of roughness values was created for each simulation based on land use information instead of using a constant roughness. This makes the input of the deep neural networks even greater and training the deep neural networks an even greater challenge. It does however make the simulations more realistic.

The data set used for training the neural network architectures in this study was relatively large leading to long training times. The training time of the convolutional encoder-decoder and modified U-net architecture was found to be in the order of 5-7 days for 10 epochs. For the recurrent convolutional LSTM (ConvLSTM) architecture, training time was found to be even longer. This was in obstacle in accessing and optimizing the hyperparameters of training and of the network. The choice for hyperparameters and the used layers in the network architecture can have a large influence on the model performance. This long training time was also the reason for using a different training scenario where only part of the training examples were used.

It was found that loading the patch examples in batches on the graphical processing unit (GPU) was time costly. Because of this, the full computing power of the GPU was not exploited. The performance of the GPU may be improved by making the input pipeline more efficient. This can be done for example by starting to load the patches of the next training step before the current training step is finished. Also the batch size and the size of the patches may lead to more efficient training.

9 Conclusion and recommendations

In this study, we explored the application of deep neural networks in a surrogate model for emulating two-dimensional (2D) hydrodynamic simulations. We proposed a framework for deep learning-based surrogate modelling of flood simulation that provides a way to generate expensive flood simulation data and train neural network architectures. A trained surrogate model may be used to create surrogate flood simulations in dike ring areas.

We used the framework to train and test three deep learning architectures. The first two architectures are feed-forward networks and the third architecture is of recurrent network type. These networks contain convolutional neural network (CNN) architectures with an encoder-decoder structure to make patch-level predictions of the flood characteristics in time. Using patches has the advantage of making a surrogate model able to create flood simulations over prone areas without restrictions on size or shape by tiling the output patches with flow predictions. Also, it allows the surrogate to focus only on regions where the flood has reached and not on the regions where no water has arrived. By making the input patches larger than the output patches and partly overlapping, the surrogate model has spatio-temporal context to be able to predict flood propagation to neighboring patches. In this way, surrogate flood simulations can be created for domains of arbitrary shape and size.

We performed various training runs of the architectures to test the influence of for example the time step, loss function and output type. In addition, we used an error reduction filter to improve the results of the surrogate model. Two feed-forward convolutional neural networks (CNNs) were trained; a convolutional encoder-decoder network and a modified U-net architecture with skip connections. We found that differences in flood characteristics between frames in a simulation (with 1 minute time step) was often very small. Predicting discrepancies in flood characteristics between the current and next time step improved the performance of the surrogate model. The convolutional encoder-decoder network showed the best results when applied to the test simulation, however this network still had difficulty to accurately predict the propagation of the flood in time.

Next, a convolutional LSTM (ConvLSTM) network was trained which is a recurrent architecture that takes as input a sequence of predicted flow patches. This architecture processes the past flood characteristics and terrain maps with two separate encoder networks and predicts the discrepancies in flood characteristics between the current and next time step. This trained network was found to be better able to predict the propagation of the flood in time. It was found that introducing memory of the flood in this recurrent architecture did improve the prediction capabilities of the surrogate. At the same time, taking into account a sequence of images increases the run time of the surrogate model as well as the training time of the network. We used this trained network architecture in a case study where the surrogate was applied to create a flood simulation in a small dike ring in the Netherlands.

We showed that the proposed framework can be used to train and apply deep learning-based surrogate models to create surrogate flood simulations for dike ring areas which may be used as a substitute of expensive hydrodynamic flood simulations. The surrogate flood simulations were found to be much more efficient in terms of calculation time compared to the hydrodynamic simulations. We trained the network architectures on fictitious flood scenario's with digital terrain model data of flood prone areas in the Netherlands and applied the trained surrogate to other locations not encountered during training. However, two major challenges in using deep neural networks inside a surrogate model for 2D flood simulations are identified. Firstly, keeping the predictions in flow characteristics accurate such that the resulting surrogate simulation is close to ground truth for long flood duration. Due to the flood propagation speed and the size of patches, the time step should be kept relatively small to accurately predict flow between patches. At the same time, floods are events that affect large areas and have long duration. The error propagation can result in large differences between a surrogate simulation and ground truth as the duration of the flood increases. Secondly, large amounts of information have to be generated from the little information present in the boundary condition. This boundary condition is specified as water depth variation in a single grid cell. Accurately generating this a lot of data from this small variation was found to be difficult for the trained surrogate models.

For further research, we provide the following recommendations based on results and findings in this study.

(1) **Avoid extreme flow conditions and instabilities in training simulations**. From the discussion of the results in Chapter 7, it was seen that some training simulations contain extremely large flow velocities and some instabilities near the breach location. The presence of such flow conditions in the training data set may not be desirable as these flow velocities are not realistic for floods resulting from levee breaches. It may be due to the definition of the boundary condition as a triangular hydrograph and the choice for a 2D boundary node in SOBEK. It is recommended to investigate how these occur and how they can be avoided as they may introduce unrealistic training examples.

(2) **Investigate the use of different boundary conditions**. In this study, a simple 2D breach node was chosen and and a simple triangular hydrograph was specified as a boundary condition. In practice, a breach model is often used that gives a more realistic simulation of the flood. A recommendation would be to investigate how such boundary conditions can be included in the surrogate modelling framework. This can improve the performance of the surrogate in practical applications.

(3) **Combine data-driven methods with simple physical models**. The difficulty with applying data-driven methods in surrogate modelling of 2D hydrodynamic simulations is that a lot of simulation has to be generated from boundary conditions which only contains a small amount of information. Combining simple physics based model, such as linearized versions of the shallow water equations with deep learning methods may be an easier problem and is worth studying.

(4) **Investigate the influence of the dimensions of the patches**. In this study, relatively large patches sizes were used. It may be easier to train a deep learning model for predicting smaller patch sizes. This can give better results in surrogate simulations and is worth investigating.

(5) **Perform hyperparameters optimization**. In this study, the influence of hyperparameters was not investigated as training time was found to be very long. Finding optimal hyperparameters and investigating the influence of using different layers in the neural network architectures can increase the performance of the surrogate model.

(6) **Study and optimize architecture layout.** It is not clear whether the model complexity has a lot of influence on the predictive performance. On a small data set, two models were compared. On this data set, the more simple model turned out to be better, however it remains unclear how much this tells us about the performance on the larger data set.

Bibliography

[1] Lionel Agostini. Exploration and prediction of fluid dynamical systems using auto-encoder technology. *Physics of Fluids*, 32(6):067103, 2020.

[2] NEM Asselman and K Heynert. Consequences of floods: 2d hydraulic simulations for the case study area central holland. *DC1-233-5*, 2003.

[3] Paul D Bates. Integrating remote sensing data with flood inundation models: how far have we got? *Hydrological processes*, 26(16):2515–2521, 2012.

[4] Steven L Brunton, Bernd R Noack, and Petros Koumoutsakos. Machine learning for fluid mechanics. *Annual Review of Fluid Mechanics*, 52:477–508, 2020.

[5] Houcine Chbab. Waterstandsverlopen rijntakken en maas. Technical report, Deltares, 2016.

[6] M Cheng, Fangxin Fang, Christopher C Pain, and IM Navon. Data-driven modelling of nonlinear spatio-temporal fluid flows using a deep convolutional generative adversarial network. *Computer Methods in Applied Mechanics and Engineering*, 365:113000, 2020.

[7] Francois Chollet et al. *Deep learning with Python*, volume 361. 2018.

[8] Karin de Bruijn and Kymo Slager. Leidraad voor het maken van overstromingssimulaties. Technical report, Deltares, 2018.

[9] JUZER F DHONDIA and GUUS S STELLING. Sobek one dimensional–two dimensional integrated hydraulic model for flood simulation–its capabilities and features explained. In *Hydroinformatics: (In 2 Volumes, with CD-ROM)*, pages 1867–1874. World Scientific, 2004.

[10] Alexander Forrester, Andras Sobester, and Andy Keane. *Engineering design via surrogate modelling: a practical guide*. John Wiley & Sons, 2008.

[11] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[12] Frauke Hoss, Sebastiaan N Jonkman, and Bob Maaskant. A comprehensive assessment of multilayered safety in flood risk management–the dordrecht case study. In *ICFM5 Secretariat at International Centre for Water Hazard Risk Management (ICHARM) and Public Works Research Institute (PWRI), Proceedings of the 5th International Conference on Flood Management (ICFMS), Tokyo, Japan*, pages 57–65, 2011.

[13] Hossein Hosseiny, Foad Nazari, Virginia Smith, and C Nataraj. A framework for modeling flood depth using a hybrid of hydraulics and machine learning. *Scientific Reports*, 10(1):1–14, 2020.

[14] R Hu, F Fang, CC Pain, and IM Navon. Rapid spatio-temporal flood prediction and uncertainty quantification using a deep learning method. *Journal of Hydrology*, 575:911–920, 2019.

[15] Delft Hydraulics. Sobek manual help. *Technical Reference Manual*, 2009.

[16] RB Jongejan, H Stefess, N Roode, W ter Horst, and B Maaskant. The vnk2 project: a detailed, large-scale quantitative flood risk analysis for the netherlands. In *Proceedings of the 5th International Conference on Flood Management (ICFM5), Tokyo-Japan*, pages 27–29, 2011.

[17] Syed Kabir, Sandhya Patidar, Xilin Xia, Qiuhua Liang, Jeffrey Neal, and Gareth Pender. A deep convolutional neural network model for rapid prediction of fluvial flood inundation. *Journal of Hydrology*, page 125481, 2020.

[18] Herman WJ Kernkamp, Arthur Van Dam, Guus S Stelling, and Erik D de Goede. Efficient scheme for the shallow water equations on unstructured grids with application to the continental shelf. *Ocean Dynamics*, 61(8):1175–1188, 2011.

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

[20] Bas Kolen, Bob Maaskant, and Frauke Hoss. Meerlaagsveiligheid: Zonder normen geen kans. *Ruimtelijke veiligheid en risicobeleid*, 2:18–25, 2010.

[21] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[22] JG Leskens, Marcela Brugnach, Arjen Y Hoekstra, and Wlatm Schuurmans. Why are decisions in flood disaster management so poorly supported by information from flood models? *Environmental modelling & software*, 53:53–61, 2014.

[23] Mario Lino, Chris Cantwell, Stathi Fotiadis, Eduardo Pignatelli, and Anil Bharath. Simulating surface wave dynamics with convolutional networks. *arXiv preprint arXiv:2012.00718*, 2020.

[24] Jonathan Masci, Ueli Meier, Dan Cireşan, and Jürgen Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. In *International conference on artificial neural networks*, pages 52–59. Springer, 2011.

[25] Bruno Merz, AH Thieken, and Martin Gocht. Flood risk mapping at the local scale: concepts and challenges. In *Flood risk management in Europe*, pages 231–251. Springer, 2007.

[26] Max D Morris and Toby J Mitchell. Exploratory designs for computational experiments. *Journal of statistical planning and inference*, 43(3):381–402, 1995.

[27] Augustus Odena, Vincent Dumoulin, and Chris Olah. Deconvolution and checkerboard artifacts. *Distill*, 1(10):e3, 2016.

[28] Saman Razavi, Bryan A Tolson, and Donald H Burn. Review of surrogate modeling in water resources. *Water Resources Research*, 48(7), 2012.

[29] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[30] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[31] Brett F Sanders. Evaluation of on-line dems for flood inundation modeling. *Advances in water resources*, 30(8):1831–1843, 2007.

[32] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting. *arXiv preprint arXiv:1506.04214*, 2015.

[33] Muhammed Sit, Bekir Z Demiray, Zhongrun Xiang, Gregory J Ewing, Yusuf Sermet, and Ibrahim Demir. A comprehensive review of deep learning applications in hydrology and water resources. *Water Science and Technology*, 2020.

[34] Jin Teng, Anthony J Jakeman, Jai Vaze, Barry FW Croke, Dushmanta Dutta, and S Kim. Flood inundation modelling: A review of methods, recent advances and uncertainty analysis. *Environmental Modelling & Software*, 90:201–216, 2017.

[35] HJ Verheij and FCM Van der Knaap. Modification breach growth model in his-om. *WL| Delft Hydraulics Q*, 3299:2002, 2002.

[36] Felipe AC Viana. Things you wanted to know about the latin hypercube design and were afraid to ask. In *10th World Congress on Structural and Multidisciplinary Optimization*, volume 19. sn, 2013.

[37] Sergiy Vorogushyn, Bruno Merz, K-E Lindenschmidt, and Heiko Apel. A new methodology for flood hazard assessment considering dike breaches. *Water resources research*, 46(8), 2010.

[38] Cornelis Boudewijn Vreugdenhil. *Numerical methods for shallow-water flow*, volume 13. Springer Science & Business Media, 2013.

[39] Christoph Wehmeyer and Frank Noé. Time-lagged autoencoders: Deep learning of slow collective variables for molecular kinetics. *The Journal of chemical physics*, 148(24):241703, 2018.

[40] MD Wilson and PM Atkinson. The use of remotely sensed land cover to derive floodplain friction coefficients for flood inundation modelling. *Hydrological Processes: An International Journal*, 21(26):3576–3586, 2007.

[41] Marcel Johannes Maria Wit and Tjerk Adriaan Buishand. *Generator of rainfall and discharge extremes (GRADE) for the Rhine and Meuse basins*. Rijkswaterstaat/RIZA, 2007.

[42] Faria T Zahura, Jonathan L Goodall, Jeffrey M Sadler, Yawen Shen, Mohamed M Morsy, and Madhur Behl. Training machine learning surrogate models from a high-fidelity physics-based model: Application for real-time street-scale flood prediction in an urban coastal community. *Water Resources Research*, 56(10): e2019WR027038, 2020.

[43] Yinhao Zhu and Nicholas Zabaras. Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification. *Journal of Computational Physics*, 366:415–447, 2018.

[44] M Zijlema. Computational modelling of flow and transport. *Collegedictaat CIE4340*, 2015.

[45] Andreas Paul Zischg, Guido Felder, Markus Mosimann, Veronika Röthlisberger, and Rolf Weingartner. Extending coupled hydrological-hydraulic model chains with a surrogate model for the estimation of flood losses. *Environmental modelling & software*, 108:174–185, 2018.

A
SOBEK simulations

A.1. Grid roughness

The bottom roughness of our tiles can be determined using land cover information, which can be constructed using geo-data files from the Key register Large-scale Topography (Basisregistratie Grootschalige Topografie, BGT, in Dutch), together with a conversion table. In addition, Buildings and Adresses, BAG (Basisregistratie Adressen en Gebouwen in Dutch), is used for location of buildings. Finally BRP (Basisregistratie Gewaspercelen) is used for land cover type of agricultural areas. All these are open data and can be downloaded through PDOK, a platform for geo datasets in the Netherlands. The conversion table from land cover to Nikuradse friction coefficient is given in Table A.1.

Table A.1: **Land type use and roughness conversion**. For a more elaborate conversion table, the reader is referred to [8].

Land use type	White Colebrook k_n [m]
Urban area, wall, quay	10
Forrest, horticulture	5
Main roads, heath, shrubland	1
Parkinglot, swamp	0.5
Grassland, sidewalk	0.2
Channels	0.05

A.2. Levee Systems

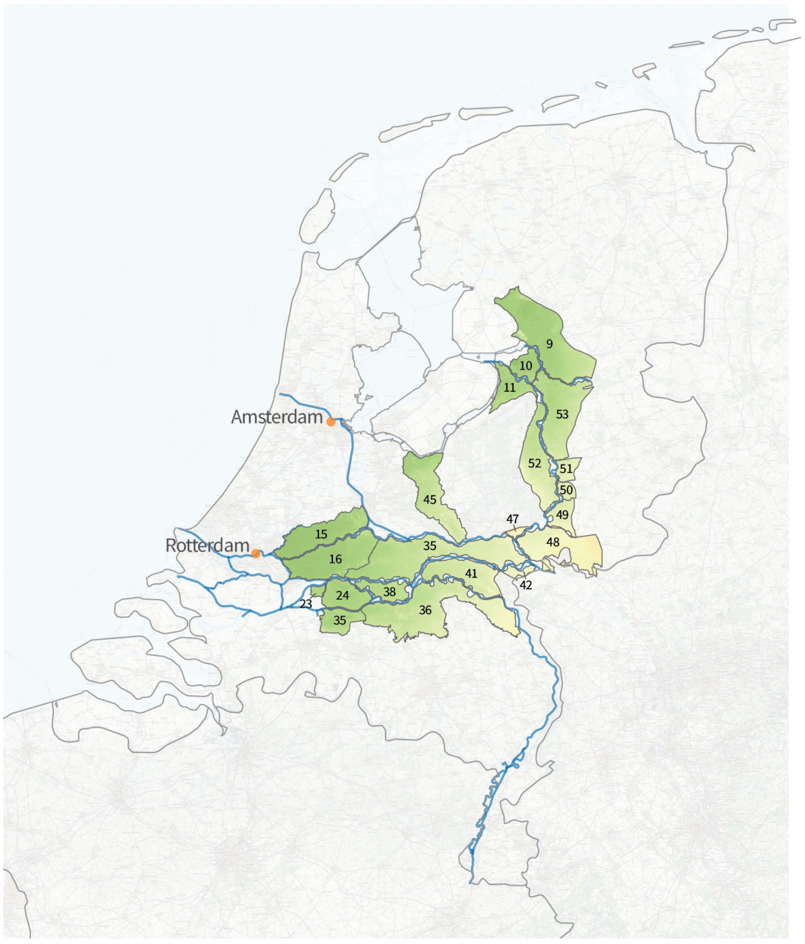


Figure A.1: Levee system with topography and roughness information used in creating data set of fictitious flood scenarios.

Num.	Name	Num.	Name
9	Vollehove	42	Ooij en Millingen
10	Mastenbroek	43	Betuwe, Tieler- en Culemborgerwaarden
11	IJsseldelta	45	Gelderse Vallei
15	Lopiker- en Krimpenerwaard	47	Arnhemse- en Velpsebroek
16	Alblasserwaard en de Vijfheerenlanden	48	Rijn en IJssel
23	Biesbosch	49	Vollehove
24	Land van Altena	50	Zutphen
35	Donge	51	Gorssel
36	Land van Heusden/de Maaskant	52	Oost Veluwe
38	Bommelerwaard	53	Salland
41	Land van Maas en Waal		

Table A.2: **Levee systems along the main rivers in the Netherlands used in this study for DEM and friction maps**. For each system, this table indicates the norm frequency, that is the return period for which the levee trajectories are designed for as well as the total area of the system.

B
Architecture details

In this chapter we provide the detailes of the different architectures used in this study. All architectures were trained usin Tensorflow and Keras software libraries.

B.1. Convolutional encoder-decoder network

Table B.1 presents the different layers used in the U-net architecture and the parameters and weights of each layer. Here, f denotes the number of feature maps and k denotes the kernel size.

B.2. Modified U-net

Table B.2 presents the different layers used in the U-net architecture and the parameters of each layer. Also the number of weights are given for the various layers. Note that the only difference with the CAE network described in previous section is the merge layers that concatenate feature maps from the encoder with feature maps in the decoder part of the network.

Layer number	Layer type	Parameters	output shape	Weights
1	Convolutional	$f = 16$, $k = 3$	(232, 232, 16)	736
2	Convolutional	$f = 16$, $k = 3$	(232, 232, 16)	2320
3	Max pooling		(116, 116, 16)	-
4	Convolutional	$f = 32$, $k = 3$	(116, 116, 32)	4640
5	Convolutional	$f = 32$, $k = 3$	(116, 116, 32)	9248
6	Max pooling		(58, 58, 32)	-
7	Convolutional	$f = 64$, $k = 3$	(58, 58, 64)	18496
8	Convolutional	$f = 64$, $k = 3$	(58, 58, 64)	36928
9	Max pooling		(29, 29, 64)	-
10	Convolutional	$f = 128$, $k = 3$	(29, 29, 128)	73856
11	Convolutional	$f = 128$, $k = 3$	(29, 29, 128)	147584
12	Upsampling		(58, 58, 128)	-
13	Convolutional	$f = 64$, $k = 3$	(58, 58, 64)	73792
14	Convolutional	$f = 64$, $k = 3$	(58, 58, 64)	73792
15	Upsampling		(116, 116, 64)	-
16	Convolutional	$f = 32$, $k = 3$	(116, 116, 32)	18464
17	Convolutional	$f = 32$, $k = 3$	(116, 116, 32)	18464
18	Upsampling		(232, 232, 32)	-
19	Convolutional	$f = 16$, $k = 3$	(232, 232, 16)	4624
20	Convolutional	$f = 16$, $k = 3$	(232, 232, 16)	4624
21	Convolutional	$f = 3$, $k = 3$	(232, 232, 3)	435
22	Cropping		(128, 128, 3)	-

Table B.1: **Convolutional encoder-decoder architecture layout**. f denotes the number of feature maps and k denotes the kernel size.

Layer type	Parameters	output shape	Weights	
1	Convolutional	$f = 16$, $k = 3$	(232, 232, 32)	1472
2	Convolutional	$f = 16$, $k = 3$	(232, 232, 32)	9248
3	Max pooling		(116, 116, 16)	-
4	Convolutional	$f = 32$, $k = 3$	(116, 116, 64)	18496
5	Convolutional	$f = 32$, $k = 3$	(116, 116, 64)	36928
6	Max pooling		(58, 58, 32)	-
7	Convolutional	$f = 64$, $k = 3$	(58, 58, 128)	73856
8	Convolutional	$f = 64$, $k = 3$	(58, 58, 128)	147584
9	Max pooling		(29, 29, 64)	-
10	Convolutional	$f = 128$, $k = 3$	(29, 29, 256)	295040
11	Convolutional	$f = 128$, $k = 3$	(29, 29, 256)	590080
12	Upsampling		(58, 58, 128)	-
13	Convolutional	$f = 64$, $k = 3$	(58, 58, 128)	295040
14	Merge			
15	Convolutional	$f = 64$, $k = 3$	(58, 58, 128)	295040
16	Upsampling		(116, 116, 64)	-

B.3. Convolutional LSTM (ConvLSTM)

The specifications of the flow encoder network and the terrain decoder network are presented in Table B.3 and Table B.4 respectively. Note that the patches with flow characteristics in the sequence share the same encoder. This flow encoder thus operates on each of the patches in the input sequence.

Layer type	Parameters	output shape	Weights	
1	Convolutional	$f = 16,\ k = 3$	(232, 232, 32)	1472
2	Convolutional	$f = 16,\ k = 3$	(232, 232, 32)	9248
3	Max pooling		(116, 116, 16)	-
4	Convolutional	$f = 32,\ k = 3$	(116, 116, 64)	18496
5	Convolutional	$f = 32,\ k = 3$	(116, 116, 64)	36928
6	Max pooling		(58, 58, 32)	-
7	Convolutional	$f = 64,\ k = 3$	(58, 58, 128)	73856
8	Convolutional	$f = 64,\ k = 3$	(58, 58, 128)	147584
9	Max pooling		(29, 29, 64)	-
10	Convolutional	$f = 128,\ k = 3$	(29, 29, 256)	295040

Table B.3: **Flow encoder network layout**. Shared encoder that operates over a sequence of past flood characteristics. f denotes the number of feature maps and k denotes the kernel size.

Layer type	Parameters	output shape	Weights	
1	Convolutional	$f = 16,\ k = 3$	(232, 232, 32)	1472
2	Convolutional	$f = 16,\ k = 3$	(232, 232, 32)	9248
3	Max pooling		(116, 116, 16)	-
4	Convolutional	$f = 32,\ k = 3$	(116, 116, 64)	18496
5	Convolutional	$f = 32,\ k = 3$	(116, 116, 64)	36928
6	Max pooling		(58, 58, 32)	-
7	Convolutional	$f = 64,\ k = 3$	(58, 58, 128)	73856
8	Convolutional	$f = 64,\ k = 3$	(58, 58, 128)	147584
9	Max pooling		(29, 29, 64)	-
10	Convolutional	$f = 128,\ k = 3$	(29, 29, 256)	295040

Table B.4: **Terrain encoder network layout**. Processes the elevation and roughness maps of a patch. f denotes the number of feature maps and k denotes the kernel size.

Layer type	Parameters	output shape	Weights	
11	Convolutional	$f = 128,\ k = 3$	(29, 29, 256)	590080
12	Upsampling		(58, 58, 128)	-
13	Convolutional	$f = 64,\ k = 3$	(58, 58, 128)	295040
14	Merge			
15	Convolutional	$f = 64,\ k = 3$	(58, 58, 128)	295040
16	Upsampling		(116, 116, 64)	-
17	Convolutional	$f = 32,\ k = 3$	(116, 116, 64)	73792
18	Merge			
19	Convolutional	$f = 32,\ k = 3$	(116, 116, 64)	73792
20	Upsampling		(232, 232, 32)	-
21	Convolutional	$f = 16,\ k = 3$	(232, 232, 32)	18464
22	Merge			
23	Convolutional	$f = 16,\ k = 3$	(232, 232, 32)	18464
24	Convolutional	$f = 3,\ k = 3$	(232, 232, 3)	435
25	Cropping		(128, 128, 3)	-

Table B.5: **Decoder network layout**. Output of the flow encoder and terrain encoder is concatenated and given as input to this decoder network. f denotes the number of feature maps and k denotes the kernel size.