Zurich University of Applied Sciences

Department Life Sciences and Facility Management

Institute of Computational Life Sciences

MASTER THESIS

Flood Modeling with Deep Learning

Author:
Eric Gericke

Supervisors:
Dr. Martin Schüle
Dr. Joao Paulo Leitao

Submitted on
July 3, 2023

Study program:
Applied Computational Life Sciences, M.Sc.

Imprint

Project: Master Thesis
Title: Flood Modeling with Deep Learning
Author: Eric Gericke
Date: July 3, 2023
Keywords: Physics, Flooding, Deep Learning, Cellular Automata

Study program:
Applied Computational Life Sciences, M.Sc.
Zurich University of Applied Sciences

Supervisor 1:
Dr. Martin Schüle
Zurich University of Applied Sciences
Email: scli@zhaw.ch
Web: Link

Supervisor 2:
Dr. Joao Paulo Leitao
Department of Urban Water Management,
EAWAG
Email: joaopaulo.leitao@eawag.ch
Web: Link

Declaration of Authorship

REMOVE THIS SECTION IF THE ORIGINAL COPY OF THE ZHAW DECLARATION OF ORIGINALITY IS USED IN THE APPENDIX.

I, Eric Gericke, declare that this thesis titled, “Flood Modeling with Deep Learning” and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at the Zurich University of Applied Sciences.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this university or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

Abstract

Flood modeling has been classically done using shallow water equations in 1 or 2D areas. Due to the computational power necessary to simulate flooding using these methods, other approaches are necessary for rapid modeling. A CA approach was introduced. Although it is significantly faster than traditional methods, it is still too slow for rapid modeling.

We evaluate a few deep learning models for the predictions of flood modeling using CADDIE2D (a CA model) for training data. Although the models perform well compared to the heurstic, they struggle to generalize to other catchment areas. The reasons for which are discussed in detail in 6.

Normalization techniques and a custom loss function for the task of linear regression for a CNN. A seperate experiment using a relatively newly proposed Neural Cellular Automaton model is also investigated for this thesis.

Acknowledgements

I would like to sincerely acknowledge the invaluable contributions of Dr. Martin Schüle and Dr. Joao Paulo Leitao throughout the completion of this masters thesis.

Dr. Martin Schüle has provided me with exceptional guidance and constant communication, serving as a reliable mentor throughout this journey. Their expertise and feedback have played a crucial role in shaping the direction of this research. I am grateful for their unwavering support and dedication to my academic growth, and the insights they have shared will always be treasured.

I would also like to express my appreciation to Dr. Joao Paulo Leitao for their assistance in providing the necessary data and their willingness to address my inquiries. Their expertise and insights have expanded my understanding of the subject matter, and I am thankful for their contribution to this thesis.

I would like to extend my gratitude to the entire research team for their collaboration and support, which have been vital to the successful completion of this work.

Lastly, I would like to thank my family, friends, and loved ones for their constant support, understanding, and encouragement throughout my academic journey. Their belief in me has been a source of motivation and strength.

To Dr. Martin Schüle, Dr. Joao Paulo Leitao, and all those who have played a part in shaping this thesis, I offer my heartfelt thanks. Your guidance, support, and insights have made a significant impact on my academic and personal growth, and I am truly grateful for your contributions.

Contents

List of Figures

Dedicated to my parents, who have supported me through out my academic journey

Chapter 1

Introduction

1.0.1 Background

Floods are one of the most common natural disasters that occur. Urban flooding is one of the key global challenges faced this century [1] as more people migrate to cities for better opportunities. Floods are often unexpected and can pose serious risks to infrastructure, economy and societies [2]. Climate change is also increasing the risk of flooding in certain areas and is resulting in higher rates of extreme rainfall in areas that have never seen these types of weather patterns. In these areas in particular, infrastructure has not been built to accommodate these types of events. In order to create useful prevention mechanisms, rapid modeling of high risk areas, preferably in real time, would be needed for adequate counter-measures to be put in place.

There are many 2D flood models have been developed to simulate urban flooding however, their complexity and computational requirements limit their applicability [3], especially when real-time predictions are required. For this reason, other models have been developed with the sole goal of reducing computational power necessary for accurate, fast predictions. One of the most successful models introduced, is a Cellular Automata based model [4]. Cellular automata are simple computer programs that only utilize local information to update cell states (like water depth in this case). However, these models are still limited in terms of computational time required. Even while running on a GPU (Graphical Processing Unit), depending on the resolution and size of the data, computational time ranges from seconds to hours.

Machine learning models have had recent success in many applications including flood modeling [2, 5, 6]. The Purpose of this thesis is to create a novel Deep learning model, by adapting a specific architecture, known as Neural Cellular Automata, proposed by Mordvintsev et al. [7] in order to predict water depths. The benefits of this approach are the low-parameter counts compared to other models, which reslutls in much faster training times as well as during inference, and the dynamics of the system can also be modeled. Another benefit of using NCA, is the potential of the model to learn the underlying physics of the system it is trying to model.

1.0.2 Related work

Data driven approaches have been studied in the flood modeling context. Many different architectures have been used such as Convolutional Neural Networks, Graph

Neural Networks, Autencoders, Deep ensembles etc. However, none have been practically applied and have not achieved widespread adoption into flood risk management, according to the author's knowledge.

One paper titled *An evaluation of deep learning models for predicting water depth evolution in urban floods* by Russo et al. [2], is reviewed as this work is very similar to this thesis and provides good benchmark models for comparison. In this paper, simulated data from the CADDIES cellular-automata flood model is used as training and testing data. The Digital Elevation Maps (DEM) used is from two catchment areas around Switzerland, named '709' and '744' (named according to a DEM numbering system). Four models were evaluated:

1. Fully Convolutional Network (FNC)
2. AutoEncoder
3. U-Net
4. Graph

Each model was trained to predict 12 time steps ahead (each time step is five minutes). They are evaluated on their unseen test set on five cells with varying water depths:

1. 0-10 cm
2. 10 - 20 cm
3. 20 - 50 cm
4. 50 - 100 cm
5. > 100 cm

The equation used for evaluation on these five buckets is the Mean Absolute error but normalized by the standard deviation of the ground truth within each bucket.

$$M_b(y_i, \hat{y}_i) = \frac{(y_i - \hat{y}_i)}{\sigma_b} \tag{1.1}$$

Where M_b is the error for a bucket; y_i is the ground truth; $\hat{y}_i$ is the predicted value; σ is the standard deviation.

They found simpler models performed better overall but no deep learning model performed significantly better than their baselines. The models used for evaluation had some major flaws. Most of the models required previous time steps as context for the model and their water depths. In the real world, this context would not be given. The only knowledge that would be available is predicted rainfall events and DEMs. There is a lot of room for improvement and this thesis uses a slightly different architecture than the ones evaluated in the above mentioned paper.

1.0.3 Objective

The overall objective for this thesis is to determine the viability of NCA's in the application of Flood modeling using only the features provided by the CADDIES model. The three main objectives of this thesis are:

1. Evaluate NCA in the application of flood modeling
2. Can the NCA model the dynamics over multiple time steps?
3. Is the model able to generalize to different rainfall events?

Chapter 2

Theoretical Background

In this chapter, the relevant literature and background information is discussed.

2.1 Cellular Automata

2.1.1 An Overview

Cellular Automata are fascinating computational models, which typically have very simple rules that result in complicated behavior. They were first introduced by John Von Neuman [8] as models for self-replicating organisms. Since then they have found relevance in an extremely wide variety of fields but mainly used in modeling biological and physical systems.

Typically, Cellular Automata are modeled on a discrete 1D or 2D lattice. The most famous Cellular Automata are John Conway's Game of Life (GOL) and Steven Wolfram's Elemetary Cellular Automata, the former of which will be discussed in the below section. From here on 'Cellular Automata' and 'Cellular Automaton' will be referred to as CA.

2.1.2 What is a CA?

CA are some of the simplest computational models that exist. They consist of a discrete lattice or grid of cells (Typically these manifolds are 1D or 2D but can be d dimensional). Each cell has a discrete state (or set of states). The most common of which, are binary states like in the case of Elementary CA and GOL [1]. Probably the most important aspect of CA systems are their local update rules'. In order for a cell to be updated, the cell considers the states of itself, as well as the states of its neighbors, and based on some rule, the cell updates. In classical CA, cells update synchronously based on some global clock [2]. Based on these features, very simple rules can result in extremely complex behaviour.
To help build intuition about CA and its associated notation, I will discuss John

[1] It is important to note, though, that variations of this exist and a lot of research has been done in the continuous state domain. A famous example of this within the CA community is Lenia: Smooth life [9]. The reason for this note is CA used for flood modeling as well as some other systems utilize this continuous state system

[2] This feature of classical CA also has variation, like in the case of stochastic CA [10], where cells update asynchronously.

Conway's Game Of Life (GOL). GOL will be mentioned a lot in this thesis because it is a simple example and widely known CA.

2.1.3 Mathematical Description of CA

Definition

An arbitrary CA is typically represented as a 4-tuple, with time, T, omitted:

$$A = (L, S, N, f) \tag{2.1}$$

Where A is the CA; L is the d-dimensional lattice; S is the set of possible states; $N \subset L$ is the neighbourhood; f is the local update rule with $f : S^N \to S$.

Defining Game Of Life

The following are the general rules written in natural language and in the subsequent section we will demonstrate the notation used for such as system.

Grid: The Game of Life is represented as a two-dimensional grid of cells. Each cell can be either alive or dead, usually represented by binary values: 1 for alive and 0 for dead. The grid is typically depicted as a matrix-like structure.

Neighbourhood: Each cell in the Game of Life has a neighborhood consisting of its eight adjacent cells, $\{-1, 0, 1\}^2$. This neighborhood is known as the Moore neighborhood. The state of a cell is influenced by the states of its neighboring cells according to the game's rules.

States: In the Game of Life, each cell can be in one of two states: alive (1) or dead (0). This binary representation simplifies the rules and allows for the emergence of interesting patterns and behaviors.

Local Update Rule: The evolution of the Game of Life is determined by a set of rules that dictate how cells change their states over time. These rules are applied simultaneously to all cells in the grid. The rules of the Game of Life are as follows:

(a) Any live cell with fewer than two live neighbors dies, as if by under population.

(b) Any live cell with two or three live neighbors survives to the next generation.

(c) Any live cell with more than three live neighbors dies, as if by overpopulation.

(d) Any dead cell with exactly three live neighbors becomes alive, as if by reproduction.

By applying these rules iteratively, the Game of Life evolves and creates various patterns, including static configurations, oscillators, and spaceships that move across the grid. The simplicity of the rules gives rise to complex and intriguing dynamics within the cellular automaton known as the Game of Life.

2.1.4 Why are they useful?

In the above section, although beautiful and impressive, the examples are not practival. But the same motivation and methods can be extrapolated to model physical or biological systems. Especially many differentiable equations that can be descritized in time and space can be modeled using CA. Many examples have been demonstrated over the years. In the following chapter 3, the CADDIE2D model is discussed in detail. But other examples include: particle simulation, chemical reactions, fire modeling, human and animal dynamics to name a few. [must find references for this.]

2.2 Deep Learning

2.2.1 What Is Deep Learning

Deep learning is a subset of techniques within a class of algorithms known as Machine Learning (ML). In general, ML algorithms power increasingly more of our day to day lives [11]. From auto-correct, to search engines, to recommendation systems, and even self-driving cars. The main difference between classical ML and DL is the former tends to require large amounts of domain knowledge and feature engineering to achieve good results. One of the main benefits of DL is their ability learn which features are important. However, classical ML tends to require a lot less data than DL models do.

DL models are derived from simple neural networks (or multilayer perceptrons), which are motivated by biological neurons. They consist of a sequence of interconnected nodes, which creates a nonlinear mapping between inputs and outputs [12]. These nodes are connected by weights and signals are created via summing the inputs to the node and passed through an activation function making the system non-linear. By doing this, it is possible for the model to approximate complicated non-linear functions. A figure of a simple multilayer perceptron is in the appendix A A.1.

There are three main categories of deep learning: Supervised learning, Unsupervised learning, and reinforcement learning. [13]. And within these three categories there are many different techniques. This thesis only considers the convolutional neural network, which is a type of supervised learning technique / architecture.

2.2.2 Convolutional Neural Networks

A Brief Overview

A Convolutional Neural Network (CNN) is a type of Artificial Neural Network (ANN) that is commonly used in image and video recognition , natural language processing, and other tasks that involve processing input data with a grid-like structure [14]. CNN's, like MLP's, are biologically motivated, however, CNN's can be said to mimic an animal's visual cortex [15]. They are able to extract important spatial features such as edges and corners of images but are also able to extract extremely complex features [16]. CNN's are much better at image-related tasks than

MLP's. One of the greatest aspects of CNN's is the kernel. The same weights are applied to whole image. This greatly reduces the parameter count compared to MLP's where every input node is connected to every output node. This allows CNN's to be trained much faster than MLP's.

Common Components of a CNN Architecture

An example of a typical CNN architecture can be seen in figure 2.2.2

Convolutional Layer: This is what makes a CNN a CNN, this layer applies a set of filters to the input data to extract relevant features for the task at hand. The filters are typically small in size and designed to detect patterns, such as edges or corners. The output of the convolutional layer then goes through a non-linear activation function, such as the rectified linear unit (ReLU) [3], which introduces non-linearity into the network.

Deepthwise Convolutional Layer: This is an interesting component and is not incredibly common. It allows for the feature mapping in specific channels. This makes it easier for the model to extract important features because it doesn't have to simultaneously learn all three channels at once [18].

Pooling Layer: This layer is meant to reduce parameter count and extract important features from the previous convolutional output. There are two types of common pooling layers:

1: Max Pooling. This applies a convolution that extracts the maximum value from the neighbourhood.

2: Average Pooling. This applies a convolution and takees the average of the neighbourhood.

1x1 Convolutional layer: This layer is very similar to a fully connected layer. It is used for dimensionality reduction or promotion. One can think of this as mini neural network for each pixel element of the previous layers output. The output of a 1x1xD convolution will have the same spatial resolution as the input into this layer with D channels.

Fully Connected Layer: This layer is just a typical MLP and is often used as the output layer for classification tasks. It works by connecting each element of the previous layer to each output node.

How Do CNN's Learn?

The first step in order to train a CNN is to employ a loss function. There are many different types of loss functions that exist. Only loss functions used for regression tasks [4] will be mentioned due to the nature of this thesis.

The first is Mean Squared Error (MSE),

[3] ReLU is generally the preferred activation function because it tends to allow for much faster training [17]

[4] A regression task involves predicting the actual values of interest apposed to a label like in classification tasks, which requires a different activation function in the output and a different loss functions.

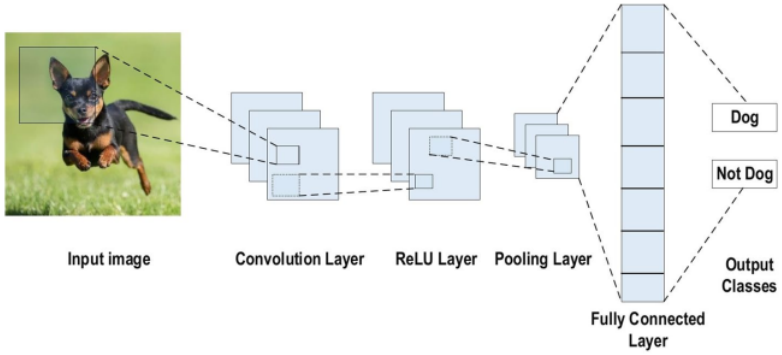

FIGURE 2.1: A simple CNN architecture taken directly from the "Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions" paper [13]

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.2}$$

The second is Mean Absolute Error (MAE),

$$MAE = \frac{\sum_{i=1}^{n}|y_i - \hat{y}_i|}{n} \tag{2.3}$$

Where n is the number of samples; y_i is the target value; $\hat{y}_i$ is the predicted value.

It is important to note that a loss function must be differentiable. The way neural networks are able to learn such complicated non-linear functions is because of the backpropagation algorithm. An optimizer algorithm like gradient descent (although many more exist) is used to minimize this loss function by driving it towards zero. It does this by calculating the optimizer with respect to the weight values for various inputs.

2.3 The Relationship Between CA and CNN

2.3.1 CNN as a CA

As it turns out, CNN's are extremely similar [5]. Cellular automata (CA) and convolutional neural networks (CNN) are both types of mathematical models that can be used to process and analyze data with a grid-like structure, such as images or time series data. [19]

Both models operate by processing the input data in a local and hierarchical manner. In a CA, the state of each cell is updated based on the states of its neighboring cells. In the case of Wolfram's Elementary CA and GOL, this is done with a look-up table. CNN's use filters / kernels, which are applied to local patches of the input data to extract relevant features. However, one can think of a neighbourhood as a NxM kernel or filter. In fact, by utilizing a cleverly constructed kernel and activation function, one can model many CA. A simple example of this would be game of life

[5] The idea for this section came from personal communication (Dr. Martin Schüle, May 2023)

represented as a convolution.

A convolution is defined as:

$$g(x,y) = \omega f(x,y)$$

Where f is the function to be convolved; ω is the kernel or filter; g is the resulting convolution. For GOL we define the kernel as

$$\omega = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 9 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

And then add an activation function, σ:

$$\sigma(x,y) = \begin{cases} 1, & \text{if } g(x,y) \in \{3, 11, 12\} \\ 0, & \text{else} \end{cases} \tag{2.4}$$

Then the components of the 4-tuple C.2 becomes,

$$A_{GOL} = (L, S, \omega, \sigma) \tag{2.5}$$

It turns out that CNN's can learn the rules of arbitrary CA's [19], for example game of life, and it doesn't need a particularly deep architecture to do so. it does this by essentially learning the kernel / filter weights that needs to be applied. To use a CNN as a CA, you essentially turn the CA into a binary classification problem to predict the state of each cell.

2.4 Neural Cellular Automata

This section attempts to summarize the paper, "Growing neural cellular automata" by Mordvintsev et al. [7]

2.4.1 Brief Background

The motivation for this NCA, like the MLP and CNN, is biologically motivated. In this case, cell morphology was the topic of interest. How do cells, using local information like the cells around them and chemical gradients 'know' what type of organ or tissue to become? This incredible ability of living systems is known as self-organization. goal of Mordvintsev et al. was to determine cell-level rules which simulate the behaviour of these biological systems. Like creating complex shapes starting from a single cell, maintaining that shape and regenerating damaged parts of the system.

2.4.2 The Model

Now that we have some basic foundational knowledge about CA and CNN's, we can move on to the real NCA (or differentiable, self-organizing systems). As discussed in section 2.3, a CNN can be seen as a type of CA, but utilizing a continuous state-space instead of a discrete one. We can also increase the amount of channels each cell has to an arbitrary number. In the paper, they found 12 hidden states to be a good number. As can be seen in (fig 2.2), this model can be thought of as a "Recurrent Residual Convolutional Network with 'per-pixel' Dropout". The 'per-pixel dropout' refers to the stochastic updating of cells. We start from a single black pixel in the center of a blank image and run the model on it for a certain number of steps where the output of the model becomes the new input (i.e. recurrent).

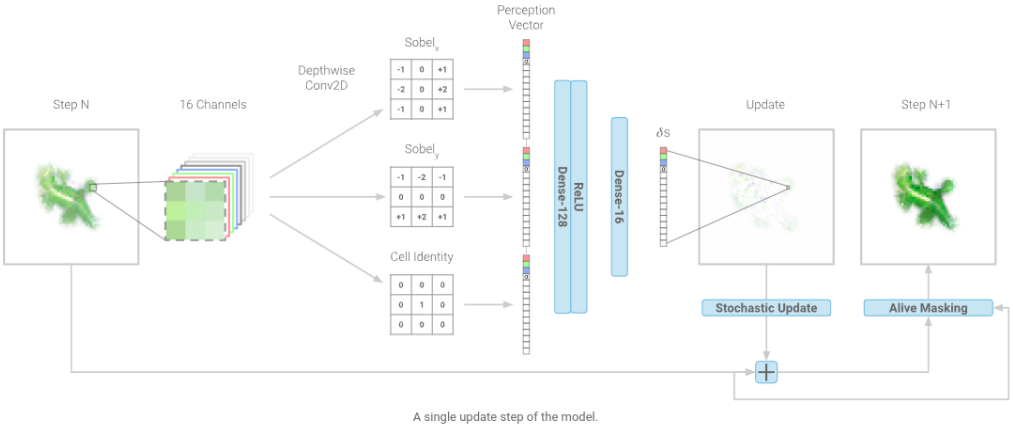

FIGURE 2.2: What a single step of the nca model looks like. This image is taken directly from the "Growing neural cellular automata" paper [7]
https://distill.pub/2020/growing-ca/

2.4.3 Training

[6] We start from a single black pixel, iterate the model over x time steps, then compare the final result of this model with the target image we are training the model. We then perform a per-pixel difference or MSE (equation 2.2) Then based on this loss we use ADAM optimizer to back-propagate through time to adjust the weights of the model until the model learns to grow or morph into the target image. An image can be seen in Appendix B (A.2)

[6]The paper outlines multiple experiments. Only the initial experiment titled "Experiment 1: Learning to Grow " will be covered.

Chapter 3

Flood Modeling

3.1 Classical Approaches

There are many different approaches to tackle flood modeling. They include Rapid flood spreading, 1D surface, 1D sewer, 1D-1D coupled, 2D surface and 1D-2D coupled [20]. The objectives of this thesis are to model the dynamics and inundation of a flood plain in 2D. Therefore only 2D modeling techniques will be mentioned.

Rapid Flood Spreading

Rapid Flood Spreading (RFS)[1], as the name implies, is a simple model to rapidly predict inundation of the flood plain. It works by equalizing the water levels of cells within a local neighbourhood. There are two stages to this process: The pre-calculation and inundation routine. In the pre-calculation routine, low points are identified in the DEM and expanded outwards. In the inundation routine, water is distributed from the inflow source to neighbouring cells using a lowest neighbour principle.

Two-Dimensional Surface

A 2D flood model represents the river network in a mesh, which provides information about the river topography and allows to define very precisely the topography of the floodplain. The hydraulics are solved using two-dimensional shallow water equations: The continuity equation (Eq. 3.1), and two-dimensional equations of motion (Eq. 3.2 and Eq. 3.3) [22].

$$\frac{\partial h}{\partial t} + \frac{\partial h V_x}{\partial x} + \frac{\partial h V_y}{\partial y} = 0 \tag{3.1}$$

$$S_{fx} = S_{ox} - \frac{\partial h}{\partial x} - \frac{V_x}{g}\frac{\partial V_x}{\partial x} - \frac{V_y}{g}\frac{\partial V_x}{\partial y} - \frac{1}{g}\frac{\partial V_x}{\partial t} \tag{3.2}$$

$$S_{fy} = S_{oy} - \frac{\partial h}{\partial y} - \frac{V_y}{g}\frac{\partial V_y}{\partial y} - \frac{V_x}{g}\frac{\partial V_y}{\partial x} - \frac{1}{g}\frac{\partial V_x}{\partial t} \tag{3.3}$$

[1] The RFS method description has been summarized from Liu and Pender [21]

Where, h is flow depth; g is gravitational acceleration; V_x and V_y are the depth-averaged velocity components along the x and y coordinates; The friction slope components S_{fx} and S_{fy} are a function of the bed slope S_{ox} and S_{oy}, pressure gradient, convective and local acceleration terms. However, the diffusive wave approximation to this equation is defined by neglecting the latter three terms.
2D surface models have the advantage of being able to model dynamics of the flood plain, duration and are very accurate.

The Problem With Classical Approaches

RFS is an extremely fast modeling scheme but comes with some major drawbacks. The can only show the final state of inundation and does not describe information about the flood. 2D surface models require fine resolution data and extremely computationally demanding. They can take hours to run and are only able to model small areas. Both of these methods have rather large drawbacks. Especially when unpredictable rainfall events occur and rapid modeling of a flood plain is required to prevent catastrophe.

3.2 The CA Approach

WCA2D) The advantage of employing CA for flood modeling is its simplicity [23] and computational speed. It has the advantage of utilizing local functions for computation. Older 2D CA models, although much faster than solving complete shallow water equations still had to solve complicated, computationally demanding equations like Manning's equation (Eq. 3.4).

The Weighted Cellular Automata

The following is an overview of the paper titled "A weighted cellular automata 2D inundation model for rapid flood analysis" by Guidolin et al. [2]. This work is an improvement on the previous CA2D model (A short literature review can be found in Appendix C). This thesis utilized data simulated by the weighted Cellular Automata 2D (WCA2D) for training and validation of deep learning models.

Ghimire et al.'s paper improved on previous CADDIES-2D model in that instead of directly solving Manning's equation for the computation of interfacial discharges between cells (Eq. 3.5), a ranking system is used and equation C.6 is used. This approach still has issues in that the ranking system equation was still solved for every time step, for each direction to limit the flow velocity. And if the computed velocity was too high, it re-calculated it. This resulted in increased computational time.

$$V = \frac{1}{n} R^{2/3} S^{1/2} \tag{3.4}$$

Where V is the cross-sectional average velocity (m/s); R is the hydraulic radius (m); S is hydraulic gradient. To calculate discharge (or volumetric flow rate), $Q = AV$ is

[2]For a detailed description on the mathematics utilized for this model I refer readers to the original paper [4]

used to rewrite Manning's equation to,

$$Q = A\frac{1}{n}R^{2/3}S^{1/2} \tag{3.5}$$

WCA2D is based on the model by Ghimire et al. but substitutes the ranking system with a weighted based system that simplifies the transition rules. This transition rule has four steps:

1. Identify the downstream neighbour cells
2. Compute the specific weight of each downstream cell based on the available storage volume
3. Compute the total amount of volume that will leave the central cell
4. for each downstream cell, set the eventual intercellular-volume which depends on the previously computed weight and total amount of volume transferred.

Overall the WCA2D model improves performance on the CA2D model but sacrifices some accuracy.

Chapter 4

Methodology

4.1 Data

The data was provided by [Eawag, institute of Aquatic Sciences]. It was generated using CADDIE2D software, which is described in Chapter 3. In the dataset, we look at four catchment areas around Switzerland, '26', '292', '709', '819' (see Fig 4.1). They are named according to a DEM naming scheme all DEM's have a spatial resolution of 1. For each DEM, there are fourteen rainfall events (see 4.2), for each rainfall event, there is a water depth map for each time step (see Fig fig:4.3, which shows spatial evolution over time), rainfall event names, a mask to remove areas outside the catchment area (if water were to propagate to this boundary, it would be considered run-off). Each time step represents 10 minutes of real time dynamics, but computational time of the CADDIES model varies.

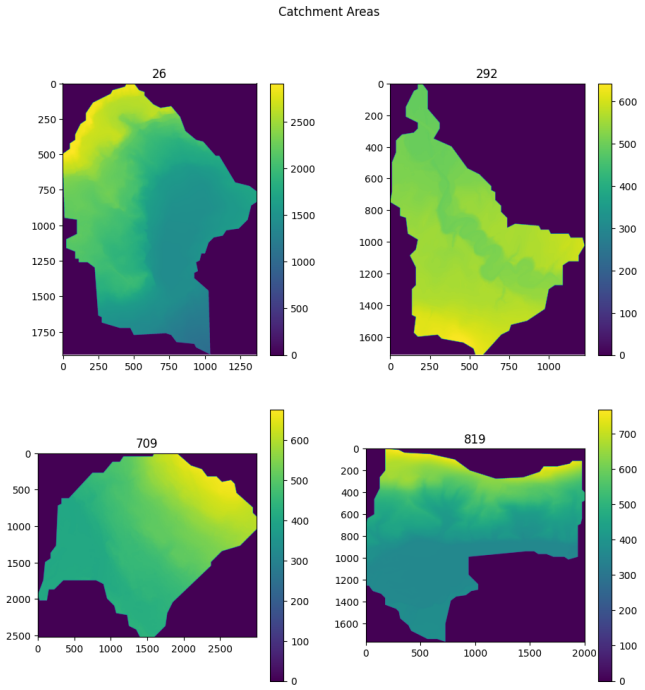

FIGURE 4.1: Elevation map for the four catchment areas of interest. Color indicates height in m; height of 0 m is outside of the catchment area

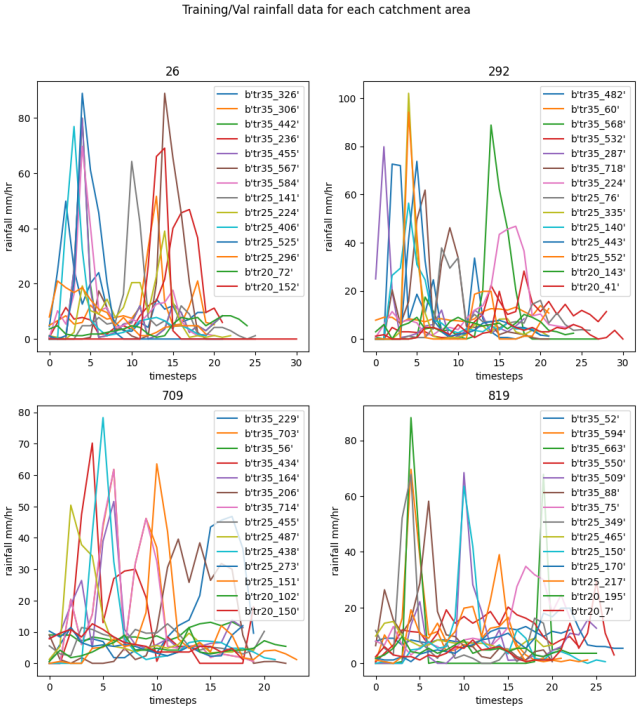

FIGURE 4.2: Rainfall events over time for each DEM

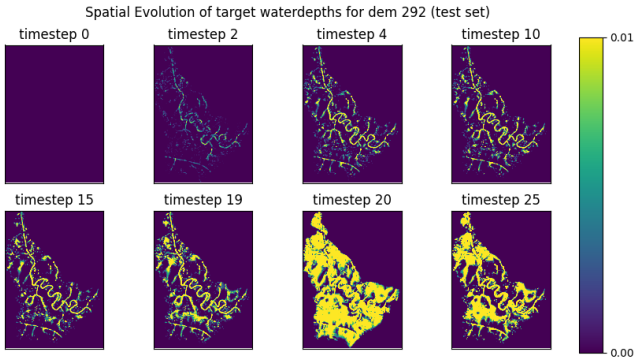

FIGURE 4.3: Spatial evolution of water depths for catchment 292. Colour indicates waterdepth (in m)

4.1.1 Data Preprocessing

Due to computational limitations, data used during training are sub-sampled to have spatial dimensions 120 x 120, while the data used for testing has spatial dimensions 400 x 400 The rainfall events, titled "rainfall _ events _ 0 - 13", are used for training and validation data (using the train _ test _ split function from the sklearn library) in a ratio of 0.8 and 0.2 respectively. The last rainfall event "rainfall _ events _ 14" is unseen and used for the test set (see Fig 4.4).
Many datasets were created but what has been mentioned above is consistent for all of them.

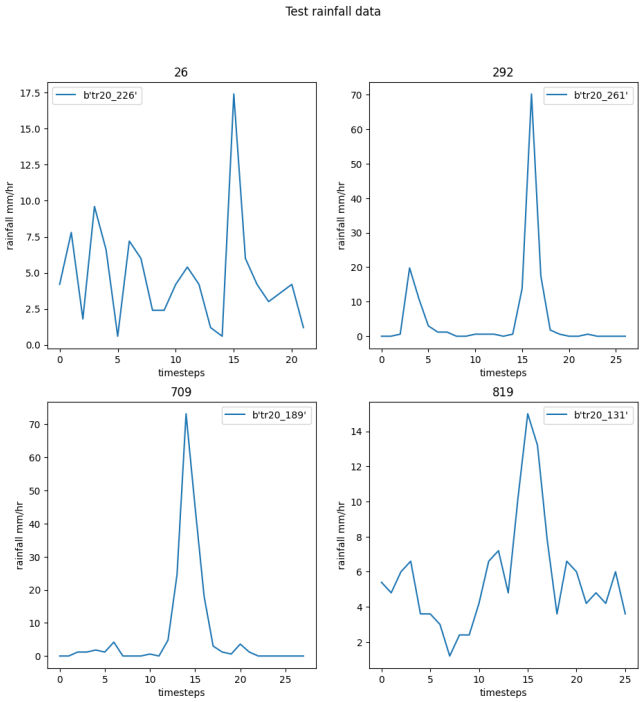

FIGURE 4.4: Rainfall events used for testing

Initial Screening

Three datasets were created for each DEM. All datasets use the same features:

1. Water depth (meters)
2. Rainfall (mm/hr)
3. DEM

A representation of the tensor can be seen in Figure 4.5

The first dataset uses the raw features generated by the CADDIES model. The second dataset uses Min Max normalization but across all features by normalizing via the DEM (see Eq. 4.1). The third uses classical min max normalization (see Eq. 4.2) for the DEM feature, rainfall is interpolated over time (see Eq 4.3), and the water depth is just divided by a constant (see Eq 4.4).

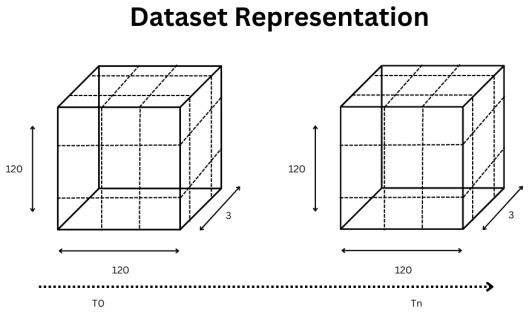

FIGURE 4.5: The shape of training data. Made using Canvas https://www.canva.com/

$$X_{norm} = \frac{X - A_{min}}{A_{max} - A_{min}} \tag{4.1}$$

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{4.2}$$

Where X is the feature to be normalized, A is the whole dataset.

$$r_{interpolated} = \frac{\frac{r}{1000}}{60} \times 10 \tag{4.3}$$

Where r is rainfall in mm/hr. The interpolated rainfall is in m/ 10 minutes.

$$X_{w\_normed} = \frac{X_w}{8} \tag{4.4}$$

Where X_w is the water depth feature. The reason to divide by 8 is to drive the water depths to a range between 0 and 1 without making the values too small. As there was a lot of variation of water depths.

Additional Features

Two additional datasets were created for evaluation. The first Adds the DEM mask to the feature list. The motivation for this is to guarantee runoff occurs outside the catchment areas. The second dataset adds rainfall directly to the water depth and we are left with only two features.

4.2 Models

Initial Screening

From previous evaluation, A simple depth-wise CNN seemed to work well. So the following model (see 4.2) was used for evaluation on the initial screening dataset:

```
    class DepthwiseCNN(tf.keras.Model):
    def __init__(self):
        super().__init__()
        self.dmodel = tf.keras.Sequential([
        tf.keras.layers.Conv2D(8, 1, activation=tf.nn.relu, padding="same"
    ),
```

```
        tf.keras.layers.DepthwiseConv2D(3, depth_multiplier = 10, activation=tf.nn.relu, padding="same"),
        tf.keras.layers.Conv2D(8, 1, activation=tf.nn.relu, padding="same"),
        tf.keras.layers.Conv2D(1, 1, activation=None, padding="same", kernel_initializer=tf.zeros_initializer)
        ])
        self(tf.zeros([1, 3, 3, 3]))


    def call(self, x):
        dx = self.dmodel(x) # Predict the difference
        f1, f2, f3 = tf.unstack(x, axis=-1) #  f1 is wd
        f1 =tf.expand_dims(f1, -1)
        return dx + f1 # return summed wd

```

Adding Complexity

Here a grid search is done using configuration files (see listing 4.2 for an example configuration) on a modified model based on listing 4.2. See listing 4.2.

```

class DepthwiseCNN(tf.keras.Model):
    def __init__(self, model_config, dset_config):
        super().__init__()
        self.dset_config = dset_config
        self.model_config = model_config
        self.dmodel = tf.keras.Sequential([
        tf.keras.layers.Conv2D(self.model_config['first1x1'], 2, activation=tf.nn.relu, padding="same"),
        tf.keras.layers.DepthwiseConv2D(3, activation=tf.nn.relu, padding="same", depth_multiplier=self.model_config['depth']),
        tf.keras.layers.Conv2D(self.model_config['second1x1'], 1, activation=tf.nn.relu, padding="same"),
        tf.keras.layers.Conv2D(1, 1, activation=None, padding="same", kernel_initializer=tf.zeros_initializer)
        ])
        self(tf.zeros([1, 3, 3, self.dset_config['num_features']]))

    def call(self, x):

        if self.model_config['diff']:
            dx = self.dmodel(x)

            if self.dset_config['mask']:
                f1, f2, f3, f4 = tf.unstack(x, axis=-1)
                f1 = tf.expand_dims(f1, -1)
                return dx + f1

            elif self.dset_config['added_rainfall']:
                f1, f2, = tf.unstack(x, axis=-1)
                f1 = tf.expand_dims(f1, -1)
                return dx + f1
            else:
                f1, f2, f3 = tf.unstack(x, axis=-1)
                f1 = tf.expand_dims(f1, -1)
                return dx + f1

        else:
            dx = self.dmodel(x)
            return dx
```

```

```

```

    model_config2 = {
        'name': 'medium_depthwise_diff',
        'depth': 8,
        'first1x1': 8,
        'second1x1': 16,
        'n_features': 3,
        'diff': True
    }
```

Inception Block Inspired

The last model looked at was an Inception Block Inspired model with varying complexity (see Listing 4.2) with an example configuration (see Listing 4.2)

```

class IncepInspired(tf.keras.Model):
    def __init__(self, model_config, dset_config):
        super().__init__()
        self.dset_config = dset_config
        self.model_config = model_config
        self.depth = tf.keras.Sequential([
        tf.keras.layers.DepthwiseConv2D(5, activation=tf.nn.relu, padding="same", depth_multiplier=self.model_config['depth']),
        tf.keras.layers.Conv2D(self.model_config['depth_1x1'], 1, activation=tf.nn.relu, padding="same")
        ])
        self.conv = tf.keras.Sequential([
        tf.keras.layers.Conv2D(self.model_config['conv_filter_1'],5 , activation=tf.nn.relu, padding="same"),
        tf.keras.layers.Conv2D(self.model_config['conv_filter_2'], 5, activation=tf.nn.relu, padding="same"),
        ])

        self.conv1x1 = tf.keras.Sequential([
        tf.keras.layers.Conv2D(self.model_config['conv_1x1'] ,1, activation=tf.nn.relu, padding="same"),
        ])

        self.dmodel = tf.keras.Sequential([
        tf.keras.layers.Conv2D(self.model_config['first_3x3'], 5, activation=tf.nn.relu, padding="same"),
        tf.keras.layers.Conv2D(self.model_config['first1x1'], 1, activation=tf.nn.relu, padding="same"),
        tf.keras.layers.Conv2D(self.model_config['second1x1'], 1, activation=tf.nn.relu, padding="same"),
        tf.keras.layers.Conv2D(1, 1, activation=None, padding="same", kernel_initializer=tf.zeros_initializer)
        ])
        self(tf.zeros([1, 3, 3, self.dset_config['num_features']]))

    def call(self, x):

        if self.model_config['diff']:
            x1 = self.depth(x)
            x2 = self.conv(x)
            x3 = self.conv1x1(x)
            y = tf.concat([x1,x2,x3], axis=-1)

            dx = self.dmodel(y)
```

```
            if self.dset_config['mask']:
                f1, f2, f3, f4 = tf.unstack(x, axis=-1)
                f1 = tf.expand_dims(f1, -1)
                return dx + f1

            elif self.dset_config['added_rainfall']:
                f1, f2, = tf.unstack(x, axis=-1)
                f1 = tf.expand_dims(f1, -1)
                return dx + f1
            else:
                f1, f2, f3 = tf.unstack(x, axis=-1)
                f1 = tf.expand_dims(f1, -1)
                return dx + f1

        else:
            x1 = self.depth(x)
            x2 = self.conv(x)
            x3 = self.conv1x1(x)
            y = tf.stack([x1,x2,x3], axis=-1)
            dx = self.dmodel(y)
            return dx
```

```
    incep_config2 = {
        'name': 'medium_incep',
        'depth': 8,
        'depth_1x1': 12,
        'conv_filter_1': 32,
        'conv_filter_2': 12,
        'conv_1x1': 12,
        'first_3x3': 32,
        'first1x1': 8,
        'second1x1': 16,
        'diff': True,

    }
```

4.3 Training

Loss Functions

MSE (see Eq. 2.2) and MAE (see Eq. 2.3) were evaluated as well as two variations of regularization (see Eq. 4.5 and Eq. 4.6) for each for a total of six loss functions.

$$R_{auto} = (W_z + W_{nz}) \tag{4.5}$$

Where Wz is the zero weight matrix calculated by summing the zeros in $\hat{y}_i$ and divided by the total number of elements. W_{nz} is the non-zero weight matrix calculated by summing the non zero elements and dividing by the total number of elements. R_{auto} was named such because it automatically adjusts the weighting based on current sub section of data.

$$R_{manual} = (W_z + W_{nz}) \tag{4.6}$$

Where Wz is the zero weight that is a chosen parameter. W_{nz} is the non zero weight that is a chosen parameter. R_{manual} is the Regularization term.

The loss functions now become,

$$Loss = (error \cdot R)\frac{1}{n} \tag{4.7}$$

where *error* is either Sum of Square Error or Absolute Error. R is either of the regularization terms in Eq. 4.5 or Eq. 4.6.

Hyper Parameter Tuning

In previous evaluation, the Adam optimizer performed the best and so was used for training in this thesis. A consistent batch size is across training. It was found to be a happy medium between amount of samples in each batch and computer memory. The learning rate for the Adam optimizer as well as number of Epochs (number of iterations through the whole dataset) was determined using a grid search method.

Along with the above, models were evaluated on predicting water difference versus predicting the water depth directly.

4.4 Evaluation

Baseline Models

Two shallow deep learning models are used for comparison. The first is a slightly shallower version of the Fully Convolutional Network used by Russo et al.

```
class BenchMark1(tf.keras.Model): # Benchmark from the evalution paper (but reduced, do to computational constraints)
    def __init__(self, model_config, dset_config):
        super().__init__()
        self.dset_config = dset_config
        self.model_config = model_config
        self.dmodel = tf.keras.Sequential([
        tf.keras.layers.Conv2D(16, 5, padding="same", activation=tf.nn.relu),
        tf.keras.layers.Conv2D(32, 5, padding="same", activation=tf.nn.relu),
        tf.keras.layers.Conv2D(32, 5, padding="same", activation=tf.nn.relu),
        tf.keras.layers.Conv2D(1, 1, activation=None)
        ])
        self(tf.zeros([1, 3, 3, self.dset_config['num_features']]))

    def call(self, x):

        if self.model_config['diff']:
            dx = self.dmodel(x)

            if self.dset_config['mask']:
                f1, f2, f3, f4 = tf.unstack(x, axis=-1)
                f1 = tf.expand_dims(f1, -1)
                return dx + f1

            elif self.dset_config['added_rainfall']:
                f1, f2, = tf.unstack(x, axis=-1)
                f1 = tf.expand_dims(f1, -1)
                return dx + f1
            else:
```

```
                f1, f2, f3 = tf.unstack(x, axis=-1)
                f1 = tf.expand_dims(f1, -1)
                return dx + f1

        else:
            dx = self.dmodel(x)
            return dx

```

The second benchmark model is a simple CNN but characterized by 1x1 convolutions in the second and third layer of the model. This model was established in prior testing.

```
    class BenchMark2(tf.keras.Model): # simple model characterized by 1x1
    convolutions
        def __init__(self, model_config, dset_config):
            super().__init__()
            self.model_config = model_config
            self.dset_config = dset_config
            self.dmodel = tf.keras.Sequential([
            tf.keras.layers.Conv2D(80, 3, padding="same", activation=tf.nn
    .relu),
            tf.keras.layers.Conv2D(64, 1, activation=tf.nn.relu),
            tf.keras.layers.Conv2D(1, 1, activation=None)
            ])
            self(tf.zeros([1, 3, 3, self.dset_config['num_features']]))

        def call(self, x):

            if self.model_config['diff']:
                dx = self.dmodel(x)

                if self.dset_config['mask']:
                    f1, f2, f3, f4 = tf.unstack(x, axis=-1)
                    f1 = tf.expand_dims(f1, -1)
                    return dx + f1

                elif self.dset_config['added_rainfall']:
                    f1, f2, = tf.unstack(x, axis=-1)
                    f1 = tf.expand_dims(f1, -1)
                    return dx + f1
                else:
                    f1, f2, f3 = tf.unstack(x, axis=-1)
                    f1 = tf.expand_dims(f1, -1)
                    return dx + f1

            else:
                dx = self.dmodel(x)
                return dx
```

And finally a simple heuristic is used for a complete comparison. The heuristic is predicting the previous time step with no change (see Eq. 4.8) [1].

$$X_w^{t+1} = X_w^{t-1} \tag{4.8}$$

Where X_w^t is the water depth at time t.

[1] The roll function from the numpy module in python was used as the heuristic. An oversight occured and instead of evaluating on no change (the original idea), the heuristic instead predicts the previous time step. What was meant to be evaluated was $X_w^{t+1} = X_w^t$

Metric

The proposed metric for evaluating the model is to use MAE (see Eq. 2.3). The goal is to minimize this as much as possible and to perform better than the baseline models and heuristic.

Chapter 5

Results

5.1 Initial Dataset Screening

DEM Screening

The Initial dataset screen was done in order to evaluate the best DEM and normalization strategy. The findings are presented here,

Loss Functions

TABLE 5.1: Final Results For Loss Functions Across DEM 292

Name	H	BM1	BM2	Model	Recurrent
MSE	0.000787	0.000914	0.002026	0.002225	0.028407
MAE	0.000787	0.001607	0.003056	0.000893	0.004070
MSE AUTO	0.000787	0.001145	0.019667	0.001983	34.082030
MAE AUTO	0.000787	0.001093	0.001836	0.000866	0.003725
MSE MAN	0.000787	0.001633	0.001352	0.000740	0.002307
MAE MAN	0.000787	0.000994	0.008186	0.000665	0.001413

Where AUTO and MAN refer to the different regularization done on MSE and MAE. H is the Heuristic, BM1 and BM2 are the benchmark models, Model is the model used during initial screening, Recurrent refers to the recurrent predictions on the test dataset.

5.2 title

Chapter 6

Discussion

6.1 Interpretation of results

Chapter 7

Conclusion

7.1 Conclusion

7.2 Outlook and future work

Appendix A

Extra Notes on Deep Learning and materials

A.1 Simple multilayer perceptron

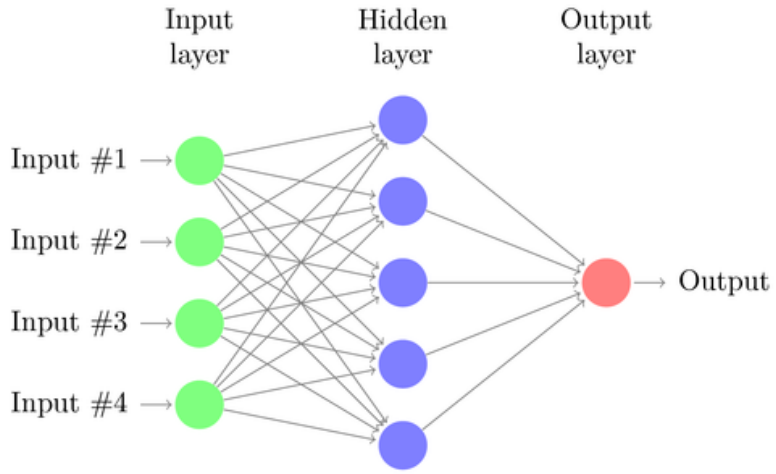

FIGURE A.1: A simple depiction of an artificial neural network
https://texample.net/tikz/examples/neural-network/

$$y = \sigma\left(\sum WX + B\right)$$

Where

y is the output or hidden layer
$\sigma$is the activation function
X is the input vector
W is the weight vector
B Bias

Where Edges are weights and nodes are the sum of the weights *inputs + bias. An activation function like sigmoid or ReLu (which performs a kind of smooth mapping) is applied to this sum. What you might you might in the above equation is its similarity with the straight line function: $y = mx + c$. It indicates that a single neuron can learn to class a binary problem as long as it is linearly separable!

A.2 Growing NCA

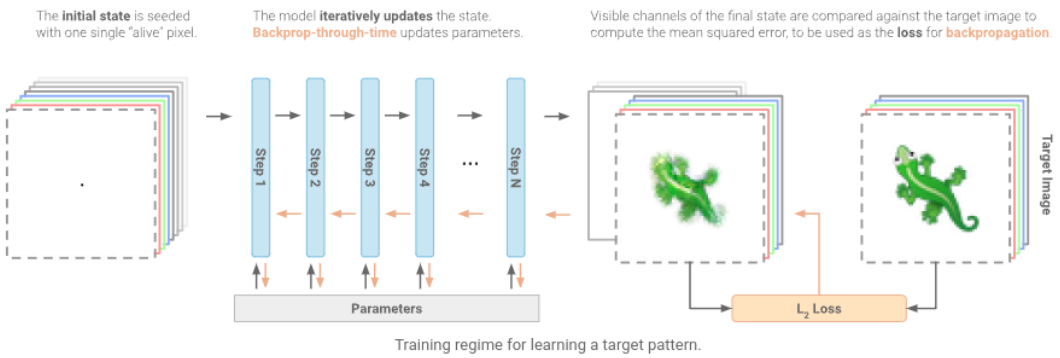

FIGURE A.2: How the model learns a target pattern. This image is taken directly the "Growing neural cellular automata" paper [7]
https://distill.pub/2020/growing-ca/

Appendix B

Building Intuition About NCA

Based on the amazing work of Mordvintsev et al. and his associated tutorials [1] , I implemented a simplified version of this NCA in Tensorflow. The first attempt was just copying the code and playing with the notebook available here. Next was simplifying the model to point I could understand the code. It turns out that a lot of the work that went into this paper was creating an incredibly robust model that was invariant to many factors, such as rotation, regrowth, and persistence. All of which I didn't really need for my purposes. And once I had a grasp of the basic concepts I could expand the model / training as needed.

Some things I did not implement from the paper:

- Damaging the model to train for regrowth
- Playing around with the fire rate as 0.5 seemed to work best based on the paper anyways
- Creating a circle mask around the image such that it doesn't rely on edges to grow certain features (or in some cases the whole image)

One method I implemented from the original paper was the pooling section.

```
class Pooling:
    def __init__(self, n_channels=16, pool_size=1024, size=(40,40),
    padding_size=16, sample_size=8):
        self.pool_size = pool_size
        self.n_channels = n_channels
        self.sample_size = sample_size
        self.size = size
        seed = Pooling.make_seed(size[0]+padding_size, size[1]+
    padding_size, self.n_channels)
        self.the_pool = seed.repeat(self.pool_size, axis=0)

    def get_sample(self):
        inds = np.random.choice(self.pool_size, self.sample_size, False)
        batch = self.the_pool[inds]
        return inds, batch

    def commit(self, inds, outputs):
        self.the_pool[inds] = outputs

```

[1] A link to his youtube channel here

This is just a way to artificially increase the amount of iterations the model performs to make sure that once the image has been accurately grown, the image remains after an arbitrary amount of time-steps. It does this by sampling final grown images from training and uses this as the initial configuration of the model. So if during training, you start from a seed configuration of a single black pixel in the center of the blank image, and allow it to grow to grow for example 50 time steps until it produces an image, then during the next iteration of training, the initial configuration will be that image and needs to remain that image for 50 time-steps. Essentially artificially increasing the time steps from 50 to 100 without having to perform back-propagation over 100 time-steps.

There were some issues with this method though. If you don't have a sufficiently large enough pool then the pool might get clogged up with terrible, failed images that are essentially just noise. it is better to use this at later stages of training when it isn't growing randomly.

Appendix C

Review of "Formulation of a fast 2D urban pluvial flood model using a cellular automata approach" by Ghimire et al.

This appendix was part of my work for track module 1 and may be helpful in understanding the updated paper.

C.1 CA Approach

There are a few problems with conventional flood modeling techniques. 1D models give us limited information about flow dynamics and 2D models are extremely computationally taxing, which limits their use. Cellular automata (CA) could be implemented to remedy this problem by reducing the computational time and cost to run these models. The focus of this review will be primarily on the CA created by [3] as the paper is very well laid-out and the CA used is well defined, albeit, complex. This CA uses regular grid cells as a discrete space for the CA setup and applies generic rules to local neighbourhood cells to simulate the progression of pluvial floods.

CA Formulation by Ghimire, et al

This model consists of five essential features of a true cellular automation: discrete space; neighbourhood (NH); cell state; discrete time step; transition rule. The square grid digital elevation model (DEM) provides the discrete space for the CA set up. A DEM is just a representation of the bare ground (bare earth) topographic surface of the Earth which excludes trees, buildings, and any other surface objects [24]. The NH used for this model consists of the central cell itself and its four cardinal adjacent cells (five cells in total). This is known as the von Neumann type NH. The Moore NH, consisting of eight surrounding cells and the central cell similar to the NH in John Conway's Game of Life, is an alternative. Precipitation occurs over the whole area of the terrain being considered. Movement of the water is mainly driven by the slopes between cells and limited by the transferrable volume and the hydraulic equations. The transferrable volume is the minimum of the total volume within the giving cell and the space available in the receiving cells. The transferrable volume is the minimum of the total volume within the giving cell and the space available

in the receiving cells. Manning's equation and the critical flow equation are applied to restrict the flow velocity. The assumption here is that water can only flow from one cell to its local NH, according to the hydraulic gradients in one computing time step. In the calculation the NH cells are ranked according to the water level, as 1 for the cell with the lowest level and 5 for the highest one, to determine the direction of flow between cells. Only the outflow fluxes (Flux as flow rate per unit area) from the central cell to its neighbours with lower ranks are calculated. Any inflow to the cells under consideration is eventually calculated as the outflow from its neighbor that has a higher water level on the opposite of the cell interface. The fluxes through the interfaces of the central cell are determined by the states of NH cells in previous time steps and stored as intermediate buffers for updating the states of cells. The states of flood depths of all cells are updated simultaneously when all interface fluxes are determined.

Main Algorithm:

Program start

1. Initialize variables –depth, water surface elevation, input(terrain, rainfall)
2. Start time loop{
3. Add precipitation depth directly to the water depth on the cells
4. Computation starts in the local NH {
   (a) Ascending cell rankings based on the water surface elevations
   (b) Layer-wise claculation of outflows from central cell
   (c) Distribution of layer-wise fluxes within the NH
   (d) Calculate the cell interfacial velocities
5. End of local NH loop }
6. Determine time step Δt required for the distriibutions appiled
7. Update simulation time: $t = t + \Delta t$
8. Update the states (depths, water surface elevation) for the new time step
9. Apply boundary conditions to suit the flow conditions
10. Data outputs for visualization and analysis
11. Repeat until the end of simulation time
12. End of time loop }

Program end

Outflow Flux Calculation

The calculation process starts with cell ranking, based on the water surface elevation in the local NH with five discrete states of cell ranks {r=1, 2, 3, 4, 5}. This is the height of each cell. Space between the water levels (water levels added on top of the height) of the cells are divided into four layers. Li ¬ is the free space between the water levels of cells ranked i and i + 1 that can accommodate the water volume from cells with higher ranks. If the rank of the central cell is r¬c (e.g. rank 3) there can be at most rc -1 number of cells receiving water as flux (because the central cell obviously can't receive water from itself) through the NH cell boundaries, if enough water is available in the central cell. Outflow volume to the layer i can be given by the following formula that is applied locally for each cell considered:

$$\Delta V_i = \min\left\{ V_c - \sum_{k=1}^{i-1} \Delta V_k, \Delta WL_i \sum_{k=1}^{i} A_k \right\} \tag{C.1}$$

Where V_c is the water volume of the central cell in the previous time step; $\Delta\ V_k$ is the volume distributed to layer k, $\sum_{k=1}^{i-1}\Delta V_k$ total volume has has been distributed to layers 1 to i-1; V_c - $\sum_{k=1}^{i-1}\Delta V_k$ represents the remaining volume available for distributing to layer i after filling i-1 layers. ΔWL_i is the water level difference between cells ranked i and i+1; $\sum_{k=1}^{i}\Delta A_k$ is the total surface area of layer i; ΔWL_i $\sum_{k=1}^{i}\Delta A_k$ is the available space for storage in layer i. For the layer adjacent to the central cell, an additional term

$$\sum_{k=1}^{i} A_k / A_c + \sum_{k=1}^{i} A_k \left(V_c - \sum_{k=1}^{i-1} \Delta V_k \right)$$

is applied to limit ΔV_i, which assumes that the water levels for all cells will reach an equivalent level. Thus, a cell with rank r receives water only from cells with higher ranks and the water received is added on top of its own water level. Thus, the total outflow flux from the central cell to a neighbouring cell ranked i is calculated as:

$$F_i = \sum_{k=i}^{r_c-1} \frac{\Delta V_k}{k} \tag{C.2}$$

For a regular grid, the areas of the central cell, A_c and the neighboring cells, A_k are constant over the domain. However, the methodology is applicable to different grid settings. Therefore, a cell containing buildings that do not allow water to flow in can be described using a variable cell area to reflect the reduced space occupied by buildings.

Depth updating

A very important step in the CA approach is the execution of the state transition rule. In the resent CA calculations, the global continuous state is the flow depth in a grid cell, which is updated for every new time step. This is done by algebraically summing the water depth from all its four neighbours. The following transition rule is used to update the flow depth:

$$d^{t+\Delta t} = d^t + \theta \frac{\sum F}{A} \tag{C.3}$$

Where θ is a non-dimensional flow relaxation parameter that can take values between 0 and 1, F is the total volume transferred to the cell under consideration as calculated from Equation C.2 and A is the cell area. The purpose of the relaxation parameter is to damp oscillations that would appear otherwise. The effect of the relaxation parameter does not impart any effect on mass conservation rather it makes the flow smooth and gradual. The values of θ are determined by numerical experiments and calibration.

Time-Step Calculation

For most 2D hydraulic modelling, higher resolution DEM data are being used, the required time steps will be shorter to ensure the stability of model computations, which often leads to large computational burden, such that many studies have been focused on reducing the computational time of simulations. The time increment, determined as the largest that satisfies the stability criteria anywhere in the whole domain, implies that for most of the cells only a fraction of the locally allowable time steps is used to integrate the solution in time. This represents a waste of computational effort and limits the use of the method. A spatially varying time step can increase solution accuracy and reduce computer run time. In this implementation we use maximum permissible velocity which ensures the minimum time steps required to distribute the applied flux. The interfacial velocity v* is determined based on the flux transferred through a cell boundary given by:

$$v^* = \frac{F}{d^* \Delta x \Delta t} \tag{C.4}$$

Where, $d*$ is the water depth of flow available at the interface, which is the difference between higher water level and higher ground elevation of the central cell and its neighbour cell to the interface

$$d* = \max\{WL_C, WL_N\} - \max\{z_C, z_N\} \tag{C.5}$$

Where, WL and z are the water levels and ground elevation respectively and the subscripts C and N represent central and neighbouring cells repectively To prevent the velocity from over shooting, a cap on the local allowable velocity is applied as given by Equation C.6 based on the Manning's formula and critical flow condition as:

$$v = \min\{\frac{1}{n} R^{\frac{2}{3}} S^{\frac{1}{2}}, \sqrt{gd}\} \tag{C.6}$$

where, the hydraulic radius R is taken to be equal to the water depth d and S is the slope of water surface elevation and is always positive for outflow calculation. If v is less than $v*$, the interfacial flux F is recalculated by replacing $v*$ with v in Equation C.4 The global time step is then calculated based on the global maximum velocity to satisfy the conventional CFL criteria. Therefore, each time the state transition rule is applied, the global time step is updated using maximum velocity calculated from all cell interfaces, as given by:

$$\Delta t = \frac{\Delta x}{\max\{v_j\}} \tag{C.7}$$

Where v_j is the velocity calculated for the jth cell interface for the entire domain.

Zürcher Hochschule
für Angewandte Wissenschaften

Life Sciences und
Facility Management

DECLARATION OF ORIGINALITY

Master's Thesis for the School of Life Sciences and Facility Management

By submitting this Master's thesis, the student attests of the fact that all the work included in the assignment is their own and was written without the help of a third party.

The student declares that all sources in the text (including Internet pages) and appendices have been correctly disclosed. This means that there has been no plagiarism, i.e. no sections have been partially or wholly taken from other texts and represented as the student's own work or included without being correctly referenced.

Any misconduct will be dealt with according to paragraphs 39 and 40 of the General Academic Regulations for Bachelor's and Master's Degree Courses at the Zurich University of Applied Sciences (dated 29 Januar 2008) and subject to the provisions for disciplinary action stipulated in the University regulations (Rahmenprüfungsordnung ZHAW (RPO)).

Town/City, Date: Signature:

…… ……..

The original signed and dated document (no copies) must be included in the appendix of the ZHAW version of all Master's theses submitted.

Bibliography

[1] Emily C O'Donnell and Colin R Thorne. "Drivers of future urban flood risk". In: *Philosophical Transactions of the Royal Society A* 378.2168 (2020), p. 20190216.

[2] Stefania Russo et al. *An evaluation of deep learning models for predicting water depth evolution in urban floods*. 2023. arXiv: 2302.10062 [cs.LG].

[3] Bidur Ghimire et al. "Formulation of a fast 2D urban pluvial flood model using a cellular automata approach". In: *Journal of Hydroinformatics* 15.3 (Dec. 2012), pp. 676–686. ISSN: 1464-7141. DOI: 10.2166/hydro.2012.245. eprint: https://iwaponline.com/jh/article-pdf/15/3/676/387056/676.pdf. URL: https://doi.org/10.2166/hydro.2012.245.

[4] Michele Guidolin et al. "A weighted cellular automata 2D inundation model for rapid flood analysis". In: *Environmental Modelling & Software* 84 (2016), pp. 378–394.

[5] Fazlul Karim et al. "A review of hydrodynamic and machine learning approaches for flood inundation modeling". In: *Water* 15.3 (2023), p. 566.

[6] Priyanka Chaudhary et al. "Flood Uncertainty Estimation Using Deep Ensembles". In: *Water* 14.19 (2022), p. 2980.

[7] Alexander Mordvintsev et al. "Growing neural cellular automata". In: *Distill* 5.2 (2020), e23.

[8] Palash Sarkar. "A brief history of cellular automata". In: *Acm computing surveys (csur)* 32.1 (2000), pp. 80–107.

[9] Bert Wang-Chak Chan. "Lenia-biology of artificial life". In: *arXiv preprint arXiv:1812.05433* (2018).

[10] Nazim Fatès. "Stochastic Cellular automata solutions to the density classification problem: when randomness helps computing". In: *Theory of Computing Systems* 53 (2013), pp. 223–242.

[11] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *nature* 521.7553 (2015), pp. 436–444.

[12] Matt W Gardner and SR Dorling. "Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences". In: *Atmospheric environment* 32.14-15 (1998), pp. 2627–2636.

[13] Laith Alzubaidi et al. "Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions". In: *Journal of big Data* 8 (2021), pp. 1–74.

[14] Zewen Li et al. "A survey of convolutional neural networks: analysis, applications, and prospects". In: *IEEE transactions on neural networks and learning systems* (2021).

[15] Christof Angermueller et al. "Deep learning for computational biology". In: *Molecular systems biology* 12.7 (2016), p. 878.

[16] Teja Kattenborn et al. "Review on Convolutional Neural Networks (CNN) in vegetation remote sensing". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 173 (2021), pp. 24–49. ISSN: 0924-2716. DOI: https://doi.

org/10.1016/j.isprsjprs.2020.12.010. URL: https://www.sciencedirect.com/science/article/pii/S0924271620303488.

[17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Communications of the ACM* 60.6 (2017), pp. 84–90.

[18] François Chollet. "Xception: Deep learning with depthwise separable convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1251–1258.

[19] William Gilpin. "Cellular automata as convolutional neural networks". In: *Phys. Rev. E* 100 (3 Sept. 2019), p. 032402. DOI: 10.1103/PhysRevE.100.032402. URL: https://link.aps.org/doi/10.1103/PhysRevE.100.032402.

[20] Dejene Tesema Bulti and Birhanu Girma Abebe. "A review of flood modeling methods for urban pluvial flood application". In: *Modeling earth systems and environment* 6 (2020), pp. 1293–1302.

[21] Yang Liu and Gareth Pender. "A new rapid flood inundation model". In: *proceedings of the first IAHR European Congress*. 2010, pp. 4–6.

[22] James S O'Brien, Pierre Y Julien, and WT Fullerton. "Two-dimensional water flood and mudflow simulation". In: *Journal of hydraulic engineering* 119.2 (1993), pp. 244–261.

[23] Stephen Wolfram. *A New Kind of Science*. English. Wolfram Media, 2002. ISBN: 1579550088. URL: https://www.wolframscience.com.

[24] Unknown. *What is a digital elevation model (DEM)?* (accessed 2022). URL: https://www.usgs.gov/faqs/what-digital-elevation-model-dem#:~:text=A%20Digital%20Elevation%20Model%20(DEM)%20is%20a%20representation%20of%20the,derived%20primarily%20from%20topographic%20maps.