Zurich University of Applied Sciences
Department Life Sciences and Facility Management
Institute of Computational Life Sciences

THESIS

Flood Modeling with Deep Learning

Author:
Eric Gericke

Supervisors:
Martin Schüle
Isaac Newton

Submitted on
June 14, 2023

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

Imprint

Project:	Thesis
Title:	Flood Modeling with Deep Learning
Author:	Eric Gericke
Date:	June 14, 2023
Keywords:	Physics, Flooding, Deep Learning, Cellular Automata
Copyright:	Zurich University of Applied Sciences

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

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

Supervisor 2:
Isaac Newton
University of Cambridge
Email: f=am@newton.com
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

Placeholder here

Contents

List of Abbreviations

CA	**C**ellular **A**utomata
NCA	**N**eural **C**ellular **A**utomata
CNN	**C**onvolutional **N**eural **N**etwork
DEM	**D**igital **E**levation **M**ap
ANN	**A**rtificial **N**eural **N**etwork
1D	**O**ne **D**imesional

For/Dedicated to/To my…

Chapter 1

Introduction

1.0.1 Background

The topic of flood modeling has become increasingly important due to the increased risk of urban flooding all over the world. Limitations of previous flood models are made increasingly apparent as the need for rapid modeling is required. Larger areas that were never at risk of flooding are now at risk. Severe, unexpected pluvial rainfall requires rapid modeling for prevension mechanisms. Unfortunately older methods just aren't fast enough. More recent CA models have allowed for faster modeling. Although these models are faster than traditional methods. They are still too slow, taking hours to model. This thesis proposes a machine learning approach to the problem of flood modeling. Deep learning has revolutionized many fields and many modeling tasks. The benefits of these models is how fast they perform computation. Often only requiring O(n) time complexity. Another feature of using deep learning approaches is that it can utilize the GPU (graphical processing unit) to further increase the speed of computation.

[Take2] Climate change is rapidly becoming more and more of a problem with strange weather phenomenon becoming more common. Sea levels are rising and heavy rainfall in some regions results in more flooding in urban environments. The need for accurate and fast flood modeling is becoming critical in order to mitigate the effects of these weather phenomenon. Classical flood modeling is slow for large areas and requires a lot of compute power. Recently, Cellular automata have been used to accurately model flooding plains, however they are also relatively slow due to the amount of times you must iterate over the data, and they still make use of shallow water equations to do so. This is where we believe the novel use of NCA could be useful. They can make use of the gpu and require far less iterations to predict water depth and potentially velocity as well. NCA have an advantage over classical CNN as it has these residual blocks which allow for more classical simulation rather than just a classification. The idea for applying machine learning methods to flood modeling isn't new. But previous approaches have had many problems, including not able to generalize to new catchment areas.

[lets try to add the two together]

1.0.2 Objective

The objectives for this thesis are as follows:

1. Normalize the data in such a way that data remains bound to physics

2. Investigate different CNN model architectures and hyper parameters for predicting water depth.

3. Create a custom loss function that constrains the model to adhere to mass conservation.

4. Can the model project arbitrarily into the future and predict multiple time steps ahead?

5. Is the trained model computationally faster than the model proposed by [1]?

Chapter 2

Methods

In this chapter, the relevant literature and background information is discussed so the reader is more easily able to follow the thesis.

2.1 Cellular Automata

2.1.1 What Are Cellular Automata

A Cellular Automaton (CA) is a system that typically consists of a discrete lattice or grid of cells. Each cell can have a discrete state, such as alive or dead, 0 or 1, etc. The cells in the grid are updated based on simple rules that depend on the cells' local neighbourhood. The entire system is typically updated simultaneously. What makes these systems fascinating is that even with very simple rules, complex emergent behavior can arise. There are two well-known CA models that I will briefly mention:

1. Wolfram's elementary CA is a one-dimensional CA with a local neighborhood of size 3, which includes the cell itself and its right and left neighbors. Some of these rules result in simple behavior, while others exhibit incredibly complex behaviors, such as the famous Rule 30 or the Turing-complete Rule 110.

2. John Conway's Game of Life is perhaps the most famous CA model. It is a two-dimensional CA on a square lattice that has been extensively studied, with new discoveries still being made to this day. This system uses the Moore's neighborhood, which is a 3x3 neighborhood that includes the central cell. The Game of Life produces incredible emergent behavior and is also Turing-complete.

2.1.2 Common notation of CA

[placeholder]

2.1.3 Why are they useful?

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

2.2 Deep Learning

2.2.1 What is deep learning

2.2.2 Optimizers for backpropogation

2.2.3 Common loss functions for regression tasks

I create this subsection because I think this needs to be carefully considered. For growing images, L2 loss is clearly the best option to reproduce images / learning to grow them. However, depending on the application of these models the loss function needs to be carefully considered and there are many to choose from. An example of this is from the paper [3, Self-Organizing Textures] where they use a L2 loss of gram-matrices by using the raw activations of VGG (Visual Geometry Group Net). Here we can also look into applying physical constraints to the model, for e.g. adhering to energy / mass conservation.

2.2.4 Convolutional Neural Network

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.

The key characteristic of a CNN is the use of convolutional layers, which apply 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 simple 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), to introduce non-linearity into the network.

In addition to convolutional layers, a CNN may also include other types of layers such as pooling layers, which reduce the spatial dimensions of the input data by selecting the maximum or average value from a set of neighboring pixels, and fully connected layers, which connect every neuron in one layer to every neuron in the next layer.

CNNs have achieved state-of-the-art performance on many computer vision tasks, such as image classification, object detection, and segmentation, and are widely used in industry and academia.

2.2.5 CNN as a CA

As it turns out, CNN's and CA's are extremely similar. 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. [2]

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, while in a CNN, filters are applied to local patches of the input data to extract relevant features. However, one can think of a neighbourhood as a NxM kernel filled with 1s. In fact, by utilizing a cleverly constructed kernel and activation function, one can model many CA's by performing this convolution and applying an activation function. A simple example of this is game of life as a convolusion:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 9 & 1 \\ 1 & 1 & 1 \end{bmatrix} \text{Kernel to be applied}$$

And activation function (or local update rule):

$$f(convolution) = \begin{cases} 1, & \text{if } convolution = 3 \text{ or } convolution = 11 \text{ or } convolution = 12 \\ 0, & \text{otherwise} \end{cases}$$

It turns out that CNN's can learn the rules of arbitrary CA's, 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.

2.2.6 Neural Cellular Automata

Now that we have some basic foundational knowledge we can move on to the real NCA (or differentiable, self-organizing systems). As discussed in the section above, a CNN can be seen as a type of CA, but utilizing a continuous state-space instead of a discrete one (as well as an arbitrary number of hidden states / channels). As described in the growing NCA paper [3], 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. Then compare the final result of this model with the target image we are training the model. We then perform a per-pixel difference loss function (L2 loss). 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' the target image from a seed state.a seed state.

Chapter 3

Flood Modeling

3.1 Classical Approaches

3.1.1 1D

A 1D flood model represents the river network by connecting cross-sections taken through the river and the land and simulating the water level as it flows through the model. It gives us a precise definition of the riverbed. The drawback to this model is it gives us limited information about the flow dynamics and the general topography of the river and flood plain [4].

3.1.2 2D

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. A drawback to this type of model is the computational time required. It is very slow is not feasible to be used over large areas. 2D models are also often linked to a 1D model of the channel between the riverbanks [4].

In a study by [5], A comparison of 1D and 2D models was made regarding Flood simulation on the Medjerda Riverin Tunisia. They found very similar information. 1D models they used (HEC RAS and MIKE 11 models) were much faster and less computationally intensive than the 2D counterpart (TELEMAC). However, the overall results were much better in the 2D model. They also found overall that the results from the 1D models and 2D models were in good agreement but was quite dependent on the accuracy of the data available to them.

3.1.3 The Problem With Classical Approaches

Although classical approaches are accurate, they are extremely computationally expensive. Solving partial differential equations is quite taxing and is limited to smaller areas at lower resolution.

3.2 CA Approach

The purpose of this review is to see how cellular automata are being implemented for flood modeling. 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 [1] 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 [6]. 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 + Δt
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:

$$\Delta V_i = \min\left\{ V_c - \sum_{k=1}^{i-1} \Delta V_k, \Delta WL_i \sum_{k=1}^{i} A_k \right\} \tag{3.1}$$

Where V_c is the water volume of the central cell in the previous time step; Δ 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; $\Delta WL_i \sum_{k=1}^{i} \Delta A_k$ 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:

$$F_i = \sum_{k=i}^{r_c-1} \frac{\Delta V_k}{k} \tag{3.2}$$

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:

$$d^{t+\Delta t} = d^t + \theta \frac{\sum F}{A} \tag{3.3}$$

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 3.2 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:

$$v^* = \frac{F}{d^* \Delta x \Delta t} \tag{3.4}$$

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

$$d* = \max\{WL_C, WL_N\} - \max\{z_C, z_N\} \tag{3.5}$$

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 localallowable velocity is applied as given by Equation 3.6 based on the Manning's formula and critical flow condition as:

$$v = \min\{\frac{1}{n}R^{\frac{2}{3}}S^{\frac{1}{2}}, \sqrt{gd}\} \tag{3.6}$$

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 3.4 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:

$$\Delta t = \frac{\Delta x}{\max\{v_j\}} \tag{3.7}$$

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

Chapter 4

Methodology

4.1 Data

Some General notes about the data:

In order to train the NCA, simulated flood data was used over a catchment area with homogenious rainfall and constant roughness coefficient. The simulated data is proven to be quite accurate and maintains mass and energy conservation. Due to a lack of compute, randomly selected subsections of the data was used and each timestep (10 or 1 minute intervals of simulation time) were used as targets for the L2 loss function.

The model works by adding the specific rainfall event directly into the water depth cell. The second channel is the DEM (digital elevation map) which is immutable in this case and the rest of the channels (25 total) are hidden channels that are not calculated in the loss function. Just like in the Growing Neural CA paper, the model is free to do what it will with these channels and only the water depth (and later hopefully velocity) will be adjusted over each iteration.

The problem I faced when slicing is there could be higher elevations with a water depth outside of this area, so when you look at the target, more water might be seen than there should be. to remidy this, I decided to use no padding and instead make the square 1 pixel in each direction larger for the input and the target is smaller. This hopefully will remedy this issue. <- this didn't work at all and I'm not entirely sure how to solve this problem now. because there is no 'run off' in the middle of the catchment area...

4.1.1 Data Acquisition

The data was provided by [Eaweg, institute of Aquatic Sciences]. It was generated using CADDIE2D software, which is discribed in detail in 3. Multiple catchment areas are part of this data. Work was done on two seperate sets of data.

4.1.2 Data Preprocessing

The data acquired is not in the correct form to be fed into the model for training. The size of the matrix is too large to fit into the local machine used in this thesis. Thefore submatrix's were created by randomly indexing into the feature map of size (50x50).

4.1.3 Features

The dataset contains the WaterDepth (in m), the DEM (digital elevation map - goes into abbreviations). Timesteps, (in seconds) and rainfall events (in mm/hour).

validation and test set

4.2 Model creation

4.2.1 Classical CNN

This was a baseline model. A generic approximation of grid search was done to find the optimial parameters.

4.2.2 Gradient Filters

A hardcoded kernel with sobel x, sobel y, and identity filters used as a way to to get gradients and information about the neighbourhood. which is then followed up with 1x1 convolutions for the computation. This filter comes straight from [3].

4.2.3 Depthwise Convolusional Layer

Instead of a 'hardcoded' filter to get gradients, the model learns should learn this filter for each feature and improves the models performance.

4.2.4 NCA Model and Adaptions

The model used is extremely similar to the model used in [3] with some minor changes.

run an 80 filter, 3x3 kernel convolution over the input to create a feature map.

Then run 1x1 convolution, 128 layer deep, with activation relu. pass that to another linear layer, output is the original channel count with no activation (although it might be possible to do something like sigmoid potentially due to the original caddieCA having a maximum allowed water transfer)

I make sure to not increment the DEM feature. I add alive masking (cell with $<$ 0.01 m water depth is set to 0, because that's how the caddieCA model works (also something not done in the paper Joao sent us.))

The output of this model becomes the new input, like a residual block.

4.2.5 Building Intuition About The Model

Based on the amazing work of [3, A. Mordvintesev et al.] and his associated tutorials, 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. 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.

4.3 Custom Loss Function

4.3.1 Custom L2 loss

Due to the nature of the data (extremely small values), the loss starts off extrememly small (example of loss here based on real values). The model also really liked to predict 0 and try to just predict the last timestep. so we create a loss function to weight the 0's according to how many 0s appear in the data. We also try to mask it so that the model cannot predict less than 0 (constraint model)

4.4 Proposed Evaluation

The proposed evaluation of the model is based on a very simple heuristic. Can the model perform better than prediciting the previous timestep? (i.e. $\delta Xt = \delta Xt - 1$)

4.5 Pipeline Creation

The creation of a pipline for training became extremely important for testing many parameters at once with different configurations for the dataset, training reigime, model creation. It allowed us to test multiple things in parrelel.

Chapter 5

Results

5.1 NCA

5.2 simple CNN

5.3 depthwise layer

5.4 performance of models on the different datasets

Chapter 6

Discussion

6.1 Interpretation of results

Chapter 7

Conclusion

7.1 Conclusion

7.2 Outlook and future work

Appendix A

Frequently Asked Questions

A.1 How do I change the colors of links?

The color of links can be changed to your liking using:

`\hypersetup{urlcolor=red}`, or

`\hypersetup{citecolor=green}`, or

`\hypersetup{allcolor=blue}`.

If you want to completely hide the links, you can use:

`\hypersetup{allcolors=.}`, or even better:

`\hypersetup{hidelinks}`.

If you want to have obvious links in the PDF but not the printed text, use:

`\hypersetup{colorlinks=false}`

A.2 How can I add a Figure in the Appendix?

You can refer to a figure in the Appendix (like A.1) and it will show up as expected.

FIGURE A.1: Bart Simpson. (2023, May 17). In Wikipedia. `https://en.wikipedia.org/wiki/Bart_Simpson`

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] 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.

[2] 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.

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

[4] Unknown. *Detailed Hydraulic (Flood) Modelling*. (accessed 2022). URL: https://www.ambiental.co.uk/services/detailed-hydraulic-flood-modelling.

[5] Mohamed Gharbi et al. “Comparison of 1D and 2D hydraulic models for floods simulation on the medjerda riverin tunisia”. In: 7 (Jan. 2016), pp. 3017–3026.

[6] 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.