**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 6, 2023

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

# Imprint

*Project:* Master Thesis
*Title*: Flood Modeling with Deep Learning
*Author*: Eric Gericke
*Date*: July 6, 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

# Abstract

Floods are one of the most frequent and catastrophic natural disasters that occur in nature. In recent years, flooding has become a major concern in urban areas around the world. Extreme rainfall events are becoming more frequent, especially in areas where these unpredictable weather phenomena have not occurred before and where infrastructure was not built to accommodate these conditions. Due to the computational time and fine resolution spatial data required to accurately model flooding behavior, there is a strong need for accurate, rapid flood modeling techniques in order for appropriate flood mitigation mechanisms to be put in place.

This thesis investigates the use of deep learning techniques, in particular neural cellular automata (NCA), in the application of flood modeling with three main objectives:

i) Evaluate NCA in the application of flood modeling

ii) Is the model able to generalize to different rainfall events?

iii) After explicitly training the model to predict water depth for a single time step, is emergent behavior observed in the form of a recurrent water depth prediction over multiple time steps.

NCA are a type of self-organizing, differentiable system that tend to have lower parameter count than other deep learning models, and are more robust to noise than typical, deterministic cellular automata (CA). In this study, simulated data from a CADDIES flood model is used as training and testing data. The Tensorflow Python library is used as a convenient framework due to the similarities between NCA and convolutional neural networks (CNN).

This thesis evaluates two NCA inspired models using a variety of training regimes and dataset constructions with differing depths. Through careful evaluation, we find that models were able to outperform the heuristic and benchmark models. However, they failed to learn meaningful behaviors to fulfill Objectives (ii) and (iii). The models typically fall into local minima immediately during training and predict no change in water depth between time steps. After careful analysis of evaluation methods, a simple model is proposed that showed much better results in relation to Objective (ii). Possible reasons for the shortcomings of the evaluation method are discussed.

# 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

# List of Tables

*Dedicated to my parents, who have shown me nothing but love and support throughout my life and 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 had no prior exposure these weather phenomena. These areas in particular, where infrastructure has not been built to accommodate these events are at much higher risk of catastrophe. 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 that have been developed to simulate urban flooding however, most of these methods require complex mathematical equations and very powerful compute. This limits their applicability [3], especially when real-time predictions are required. For this reason, other models have been developed with the sole goal of reducing the computational power necessary for accurate, fast predictions. One of the most successful models introduced, is a Cellular Automata based approach [4]. Cellular automata are simple computer programs that only utilize local information to update cell states (like water depth and velocity 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 was to create a novel Deep learning model, by adapting a specific architecture, known as Neural Cellular Automata (NCA), proposed by Mordvintsev et al. [7] in order to predict water depth over a flood plain and model the spatial evolution over time. The benefits of this approach are the low parameter counts compared to other models, which results in much faster training and inference. NCA also have the benefit over other deep learning models due to their self organizing nature. 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 model is used for training and testing the deep learning models. 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; $\sigma$ 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. This will be discussed in more detail in Chapter 4.

### 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 a subset of features provided to the CADDIES model. The three main objectives of this thesis are as follows:

i) Evaluate NCA in the application of flood modeling

ii) Is the model able to generalize to different rainfall events?

iii) After explicitly training the model to predict water depth for a single time step, is emergent behavior observed in the form of a recurrent water depth prediction over multiple time steps.

# Chapter 2

# Theoretical Background

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

## 2.1 Cellular Automata

### 2.1.1 An Overview

CA are fascinating mathematical 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 a wide variety of fields but are mainly used for modeling biological and physical systems.

Typically, CA are modeled on a discrete 1D or 2D lattice. The most famous CA are John Conway's Game of Life (GOL) and Steven Wolfram's Elementary CA, the former of which will be discussed in the below section.

### 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 predefined 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. Including Turing Completeness (like in the case of GOL and Rule 110)

To help build intuition about CA and its associated notation, I will discuss GOL.

[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 artificial life (ALIFE) 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.

### 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 neighborhood; $f$ 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.

To describe GOL mathematically using the above rules we define $A_{GoL} = (L, S, N, \phi)$, where $L = Z^2$ is the two-dimensional discrete lattice, $S = \{0, 1\}$ is the singular state set, and $N = \{-1, 0, 1\}^2$ is the Moore neighborhood (or the eight surrounding neighborhood and itself). We say the neighborhood sum of some site $x$ is:

$$S^t(x) = \sum_{n \in N} A^t(x + n) \tag{2.2}$$

Every site is synchronously updated according to the local rule:

$$A^{t+1} = \begin{cases} 1 \text{ if } A^t(x) = 0 \text{ and } S^t(x) \in \{3\} \\ 1 \text{ if } A^t(x) = 1 \text{ and } S^t(x) \in \{2,3\} \\ 0 \text{ otherwise} \end{cases} \tag{2.3}$$

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 in an organized structure. The simplicity of these rules gives rise to complex and intriguing dynamics (see Fig 2.1).

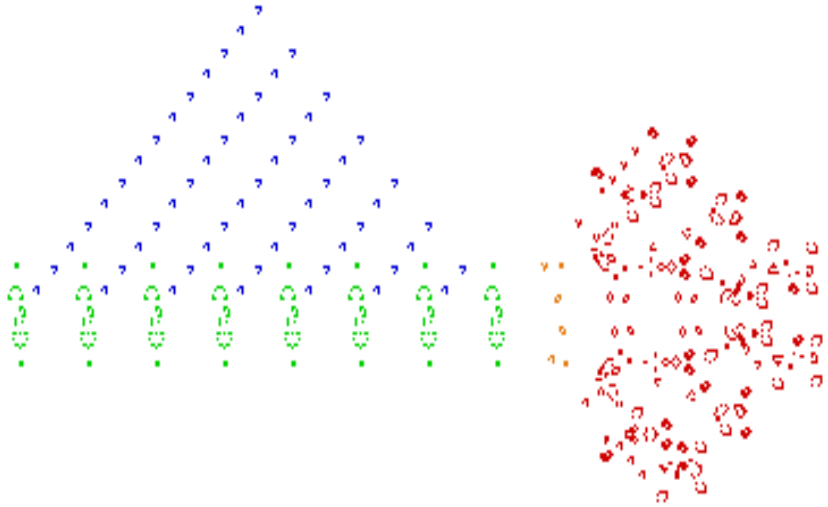

FIGURE 2.1: The Breeder Pattern in Game of Life. Created by Hyperdeath - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=5117698

### 2.1.4 Why are they useful?

In the above section, although beautiful and impressive, the examples are not practical. But the same motivation and methods can be extrapolated to model physical or biological systems. For example, many differentiable equations that can be discretized in time and space can be modeled using CA. Many examples have been demonstrated over the years. In the following Chapter 3, the CADDIE model is discussed in detail. Other examples of practical CA include: particle simulation, chemical reactions, fire modeling, human and animal dynamics [11–14] to name a few.

## 2.2 Deep Learning

### 2.2.1 What Is Deep Learning

Deep learning (DL) 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 [15]. From auto-correct [16], to search and recommendation systems [17], and even self-driving cars [18]. 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 [19]. 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 [20]. 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.

**Simple Multilayer Perceptron**

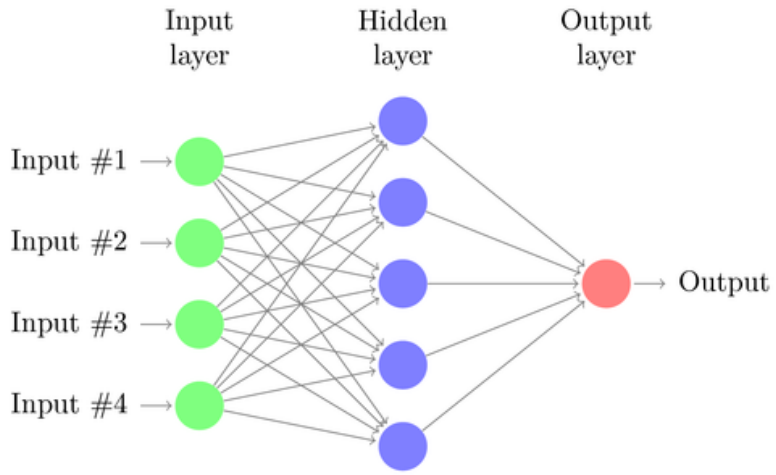


FIGURE 2.2: A simple depiction of an artificial neural network
https://texample.net/tikz/examples/neural-network/

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

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 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!.

#### Categories of Deep Learning

There are three main categories of deep learning: Supervised learning, Unsupervised learning, and reinforcement learning. [21]. 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 [22]. CNN's, like MLP's, are biologically motivated, however, CNN's can be said to mimic an animal's visual cortex [23]. They are able to extract important spatial features such as edges and corners of images but are also able to extract extremely complex features [24]. 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 layer allows for feature mapping in singular channels. This makes it easier for the model to extract important features because it doesn't have to simultaneously learn all three channels (in the case of images) at once [26].

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 takes 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

[3] ReLU is generally the preferred activation function because it tends to allow for much faster training [25]

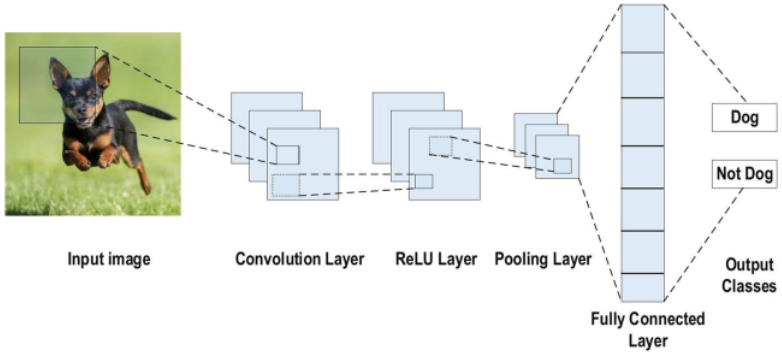


FIGURE 2.3: A simple CNN architecture taken directly from the "Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions" paper [21]

miniature 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),

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.4}$$

The second is Mean Absolute Error (MAE),

$$MAE = \frac{\sum_{i=1}^{n}|y_i - \hat{y}_i|}{n} \tag{2.5}$$

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.

[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.

## 2.3 The Relationship Between CA and CNN

### 2.3.1 CNN as a CA

As it turns out, CNN's are extremely similar to conventional CA [5]. CA and CNN's 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. [27]

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 represented as a convolution.

A convolution is defined as:

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

Where $f$ is the function to be convolved; $\omega$ 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, $\sigma$:

$$\sigma(x,y) = \begin{cases} 1, & \text{if } g(x,y) \in \{3, 11, 12\} \\ 0, & \text{else} \end{cases} \tag{2.6}$$

Then the components of the 4-tuple (see Eq. A.2) becomes,

$$A_{GOL} = (L, S, \omega, \sigma) \tag{2.7}$$

It turns out that CNN's can learn the rules of arbitrary CA's [27], for example GOL, 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 (with a binary state space), you essentially turn the CA into a binary classification problem to predict the state of each cell.

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

## 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. The goal of Mordvintsev et al. was to determine cell-level rules which simulate the behavior of these biological systems, like creating complex shapes starting from a single cell, maintaining that shape and regenerating itself once damaged. This is resistance to perturbation is a key difference between NCA and CA. Typical, deterministic CA are extremely sensitive to noise. A single cell state change may destroy the whole system and result in chaos, where NCA can learn to adapt.

### 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 2.4), 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, where only a certain percentage of cells are updated to the next time step. We start from a single black pixel in the center of a blank image and run the model on it over a number of steps where the output of the model increments the input (i.e. recurrent).

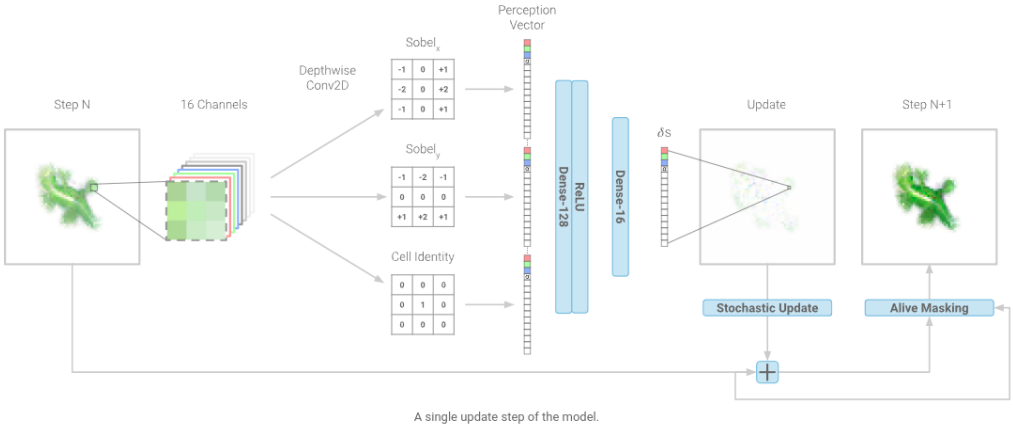


FIGURE 2.4: What a single step of the nca model looks like. This image is taken directly from the "Growing neural cellular automata" paper [7]
https://distill.pub/2020/growing-ca/

### 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 2.4). Based on this loss, we use the ADAM optimizer to back-propagate through time and adjust the weights accordingly until the model learns to grow or morph into the target image (see Fig 2.5).

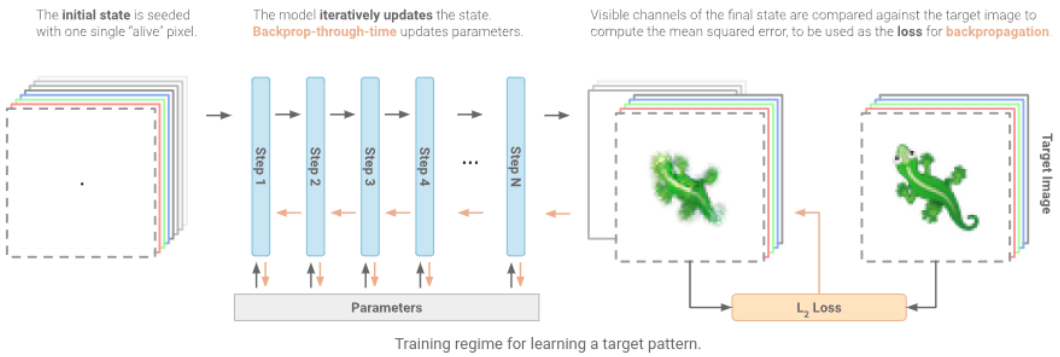


FIGURE 2.5: How the model learns a target pattern. This image is taken directly the "Growing neural cellular automata" paper [7] https://distill.pub/2020/growing-ca/

[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 [28]. 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 neighborhood. There are two stages to this process: The first is the pre-calculation, where low points are identified in the DEM and expanded outwards. Secondly, the inundation routine, water is distributed from the inflow source to neighboring cells using a lowest neighbor 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 [30] (Eq. 3.2 and Eq. 3.3) .

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

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

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

[1]The RFS method description has been summarized from Liu and Pender [29]

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. They can only show the final state of inundation and does not describe information about the flood. 2D surface models require fine resolution data and are 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

The advantage of employing CA for flood modeling is its simplicity [31] 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 A). 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 A.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 neighbor 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 and have a spatial resolution of 1 m. 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 4.3, which shows spatial evolution over time). The dataset provided also contains rainfall event names and finally, 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, but computational time of the CADDIES model varies for each time step computed.

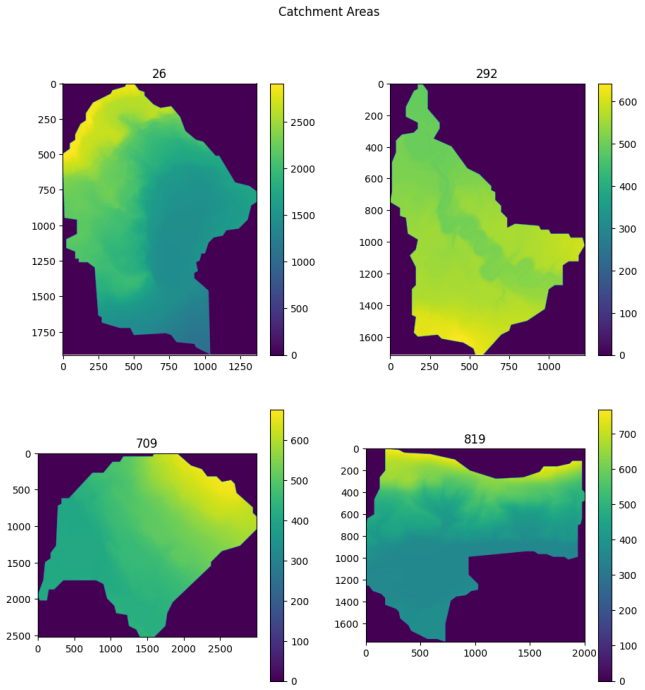


FIGURE 4.1: Elevation map for the four catchment areas of interest. Color indicates height in m; height of 0 m is outside of the catchment area

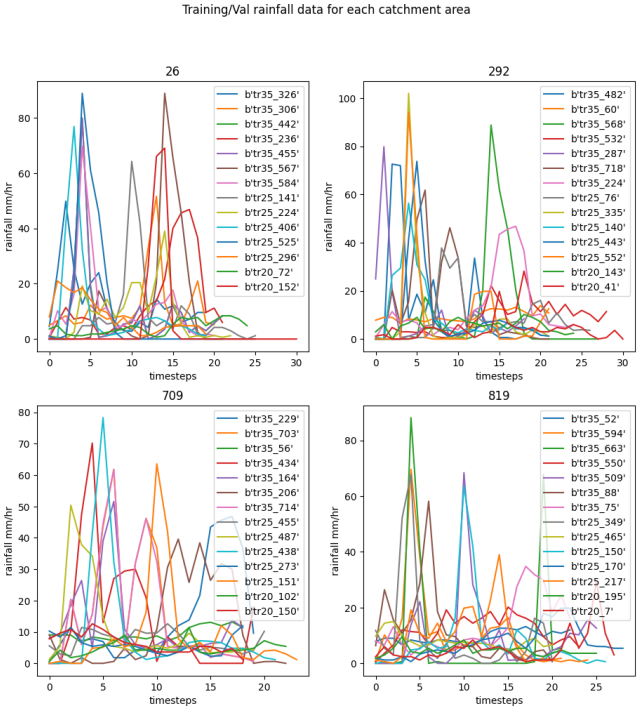


FIGURE 4.2: Rainfall events over time for each DEM

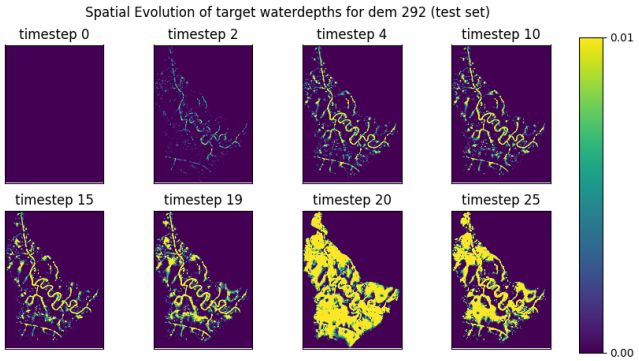


FIGURE 4.3: Spatial evolution of water depths for catchment 292. Colour indicates waterdepth (in m)

### 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.

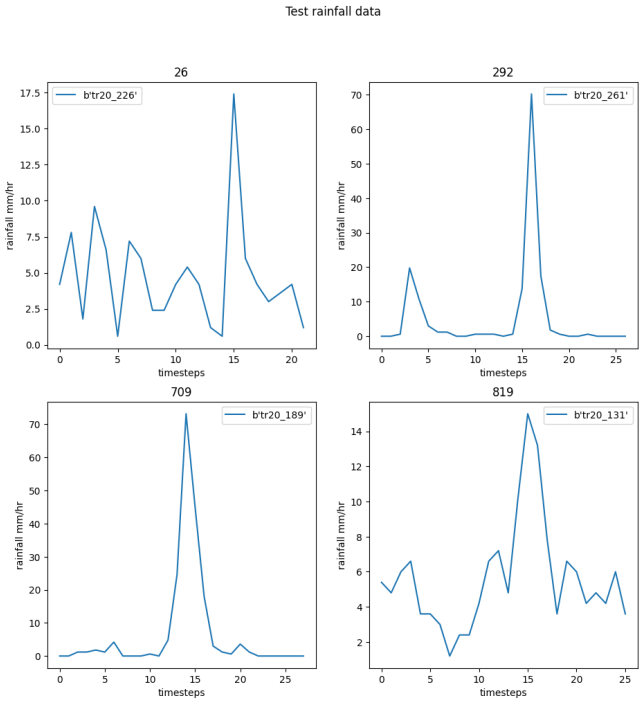


FIGURE 4.4: Rainfall events used for testing

#### 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).

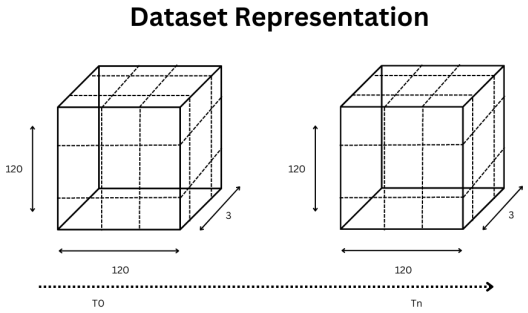


FIGURE 4.5: The shape of training data. Made using Canvas https://www.canva.com/

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

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

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 due to large variation in 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

```

The initial screening model has the following model summary,

```
Model: "Inital Screening"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
sequential (Sequential)      (1, 3, 3, 1)              1489
=================================================================
Total params: 1,489
Trainable params: 1,489
Non-trainable params: 0
_________________________________________________________________
None
```

### 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
    }
```

Three models are looked at. Shallow, Medium, and Complex models. Their model summaries, respectively, are,

```
Model: "depthwise_shallow"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
sequential_2 (Sequential)    (1, 3, 3, 1)              265
=================================================================
Total params: 265
Trainable params: 265
Non-trainable params: 0
_________________________________________________________________
None

Model: "depthwise_medium"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
sequential_3 (Sequential)    (1, 3, 3, 1)              1801
=================================================================
Total params: 1,801
Trainable params: 1,801
Non-trainable params: 0
_________________________________________________________________
None

Model: "depthwise_complex"
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
sequential_4 (Sequential)    (1, 3, 3, 1)              23913
=================================================================
Total params: 23,913
Trainable params: 23,913
Non-trainable params: 0
_________________________________________________________________
None
```

### 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). The use of this architecture is motivated by the fact that very few features are used as input and the inception block architecture is enrich the feature space through the use of sub networks [32]

```

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,

    }
```

Two versions of the model are created: Simple and Medium [1]. They have the following model summaries,

```
Model: "incep_inspired_simple"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
sequential_5 (Sequential)    (1, 3, 3, 8)              110
_________________________________________________________________
sequential_6 (Sequential)    (1, 3, 3, 8)              1388
_________________________________________________________________
sequential_7 (Sequential)    (1, 3, 3, 8)              32
_________________________________________________________________
sequential_8 (Sequential)    (1, 3, 3, 1)              5041
```

[1] The complex model was not able to train properly on the authors computer

```
=================================================================
Total params: 6,571
Trainable params: 6,571
Non-trainable params: 0
_________________________________________________________________
None

Model: "incep_inspired_medium"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
sequential_9 (Sequential)    (1, 3, 3, 12)             924
_________________________________________________________________
sequential_10 (Sequential)   (1, 3, 3, 12)             12044
_________________________________________________________________
sequential_11 (Sequential)   (1, 3, 3, 12)             48
_________________________________________________________________
sequential_12 (Sequential)   (1, 3, 3, 1)              29257
=================================================================
Total params: 42,273
Trainable params: 42,273
Non-trainable params: 0
_________________________________________________________________
None
```

## 4.3 Training

**Loss Functions**

MSE (see Eq. 2.4) and MAE (see Eq. 2.5) 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 above baseline model has a model summary,

```
Model: "bench_mark1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
sequential (Sequential)      (1, 3, 3, 1)              39713
=================================================================
Total params: 39,713
Trainable params: 39,713
Non-trainable params: 0
_________________________________________________________________
None
```

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
```

It has a model summary of,

```
Model: "bench_mark2"
_________________________________________________________________
```

```
Layer (type)                 Output Shape              Param #
=================================================================
sequential_1 (Sequential)    (1, 3, 3, 1)              7489
=================================================================
Total params: 7,489
Trainable params: 7,489
Non-trainable params: 0
_________________________________________________________________
None
```

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) [2]. Evaluation is done on all the baseline models for predicting a single time step ahead (10 minutes). But a recurrent version is also tested. After training the models, The output of the model becomes the new input. This is done for all time steps in the test data set.

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

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

### Metric

The proposed metric for evaluating the model is to use MAE (see Eq. 2.5). The goal is to minimize this as much as possible and to perform better than the baseline models and heuristic.

[2]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 $X_w^{t+1} = X_w^t$

# Chapter 5

# Results and Discussion

All results shown in this section come from data analysis in Appendix B. All code and configuration files used to generate these results can be found here : `https://github.com/afrodisac/FloodModelingThesisCode`

## 5.1 Initial Dataset Screening

### DEM Screening

The Initial dataset screen was done in order to evaluate the best DEM and normalization strategy. [1]. The plots and figures, unless otherwise specified, are evaluated using classic MAE (see Eq. 2.5).

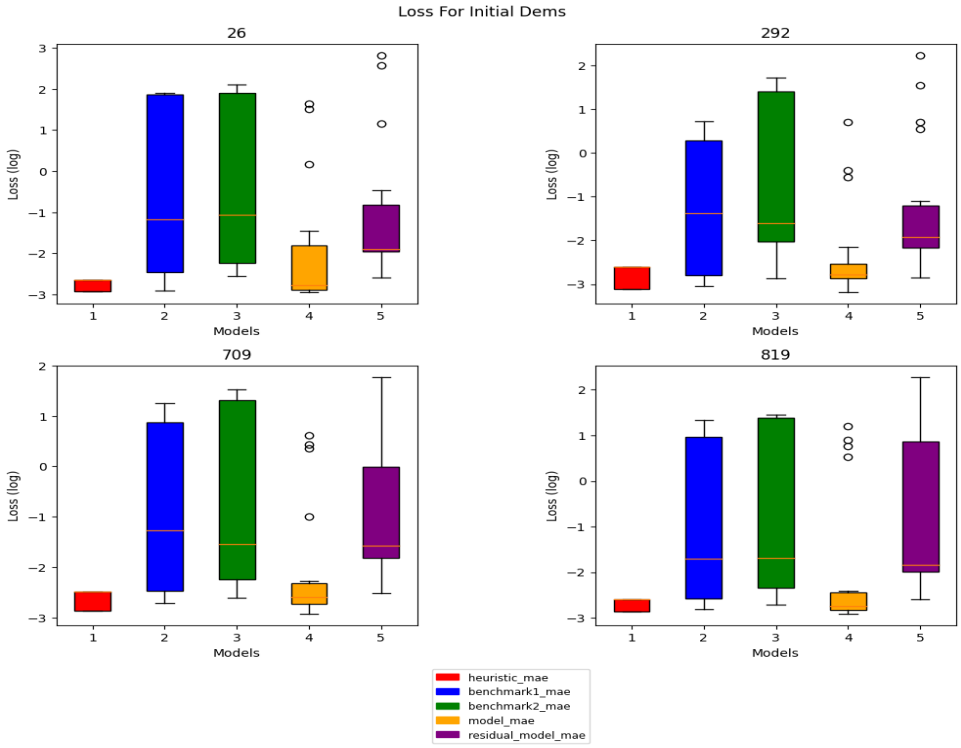


FIGURE 5.1: Loss across all models evaluated over the initial screening datasets

[1] In the legend, you will see a label "residual _ model _ mae". This is a typo and should be called Recurrent mae

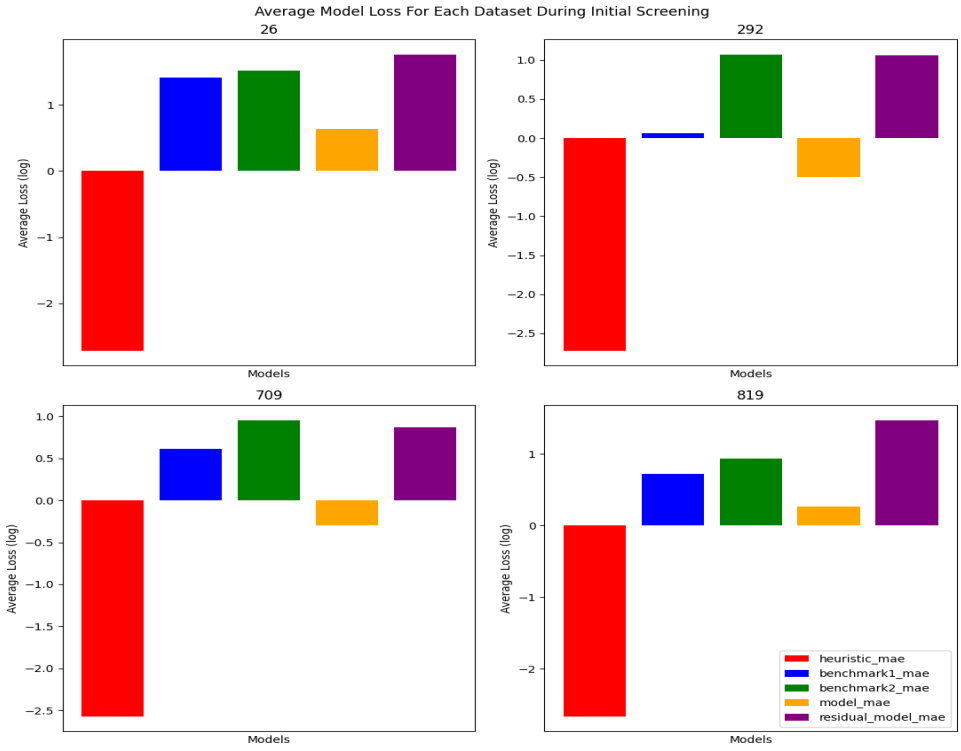


FIGURE 5.2: Mean loss across all models evaluated over all normalization strategies

Fig. 5.1 shows a box plot for all DEMS. It includes all the normalization strategies (see 4.1.1). The model used is the DepthwiseCNN model (see 4.2) Fig. 5.2 is a bar plot that shows the mean across all DEMs, normalization strategies, and loss functions (see loss functions defined 4.3). This was the first test performed and was done in order to determine which dataset to move forward with due to computational limitations and time required for training. Fig. 5.1 shows some interesting variation across the datasets. DEM 292 and 709 seemed to show the most promise, but the final decision was to move forward with DEM 292. When we look at the mean values (see Fig. 5.2), We see that DEM 292 has a slightly lower loss than DEM 709. This decision was relatively arbitrary, however, it did have the lowest loss model and showed the fewest outliers. Also the recurrent time step predictions were also the lowest.

### Normalization strategy

Fig. 5.3 shows the loss for each normalization strategy across loss functions. This was used to determine which normalization performed the best overall.

The decision for which normalization strategy to use was a challenging task. And the type of normalization techniques may have positive or negative effects during training [33]. Due to the nature of the data, where DEM values are extremely large, rainfall values and water depths have a huge range and no real limit. Fig. 5.3 shows the losses for different normalization strategies. The raw, un normalized data, interestingly, showed a very low loss for the initial model but the recurrent prediction was quite poor and the benchmark models performed the worst overall. The "all Normed" dataset where all features are normalized according to the DEM map showed the greatest variation and seems to be ultimately unpredictable. This leaves

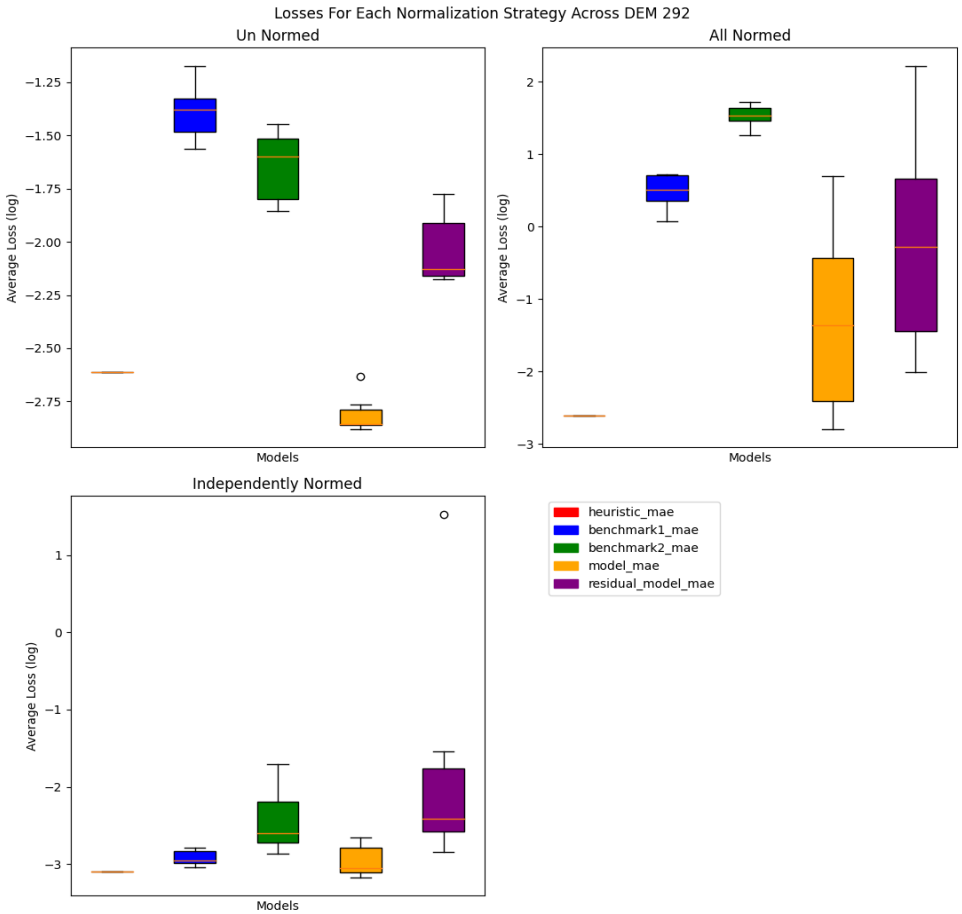


FIGURE 5.3: Box plot showing the loss for different normalization strategies for DEM 292

us with the independently normalized dataset which shows the least variation and lowest losses. Moving forward, the Independently Normalized dataset is used.

**Loss Functions**

Table 5.1 Hows the loss of all loss functions used for testing over DEM 292. The Column Names are Heuristic (H), Benchmark 1 (BM1), Benchmark 2 (BM2), The current model in question (In this case the initial screening model), and the model but using recurrent prediction.

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. Fig 5.4 shows this result visually in bar plot. These results determine which loss function to use for the rest of the evaluation.

Choosing the right loss function is a critical task for DL. The problems with classical MAE and MSE, which are essentially the only options for regression tasks like this (when dealing with two-dimensional, multi-channel tensors), is the bias they introduce towards predicting 0 change in water depth. The nature of datasets like this, is its unbalanced nature. Looking at the rainfall events (see Fig. 4.2 and 4.4), we see there are many moments where there is very small or no rainfall. On top of that, the simulated data thresholds water depth. When water depth is less than 0.01 m, it outputs 0 (see Eq. 5.1).

$$Wd_{new} = \begin{cases} 0, & \text{if } Wd < 0.01 \\ Wd, & \text{Otherwise} \end{cases} \tag{5.1}$$

Table 5.1 shows the loss across models for each loss function variation across the independently normalized dataset for DEM 292 (see Fig 5.4) We see that MSE loss doesn't generally perform as well as MAE. This might be because the metric chosen was mae, so there is potential for an injected bias here. Weights chosen for the Manual weighted loss functions were 0.8 zero weight and 0.2 non zero weight. This may seem counter intuitive to what is mentioned above but it still performed the best. This is because predicting 0 change in water depth outperformed the Heuristic. MSE and MSE AUTO variations performed the worst by a large margin. While Manual MAE performed the best. Therefore, the Manual MAE variation was chosen to for further investigation. This is an interesting finding because MSE is typically the preferred loss function for regression tasks [34].

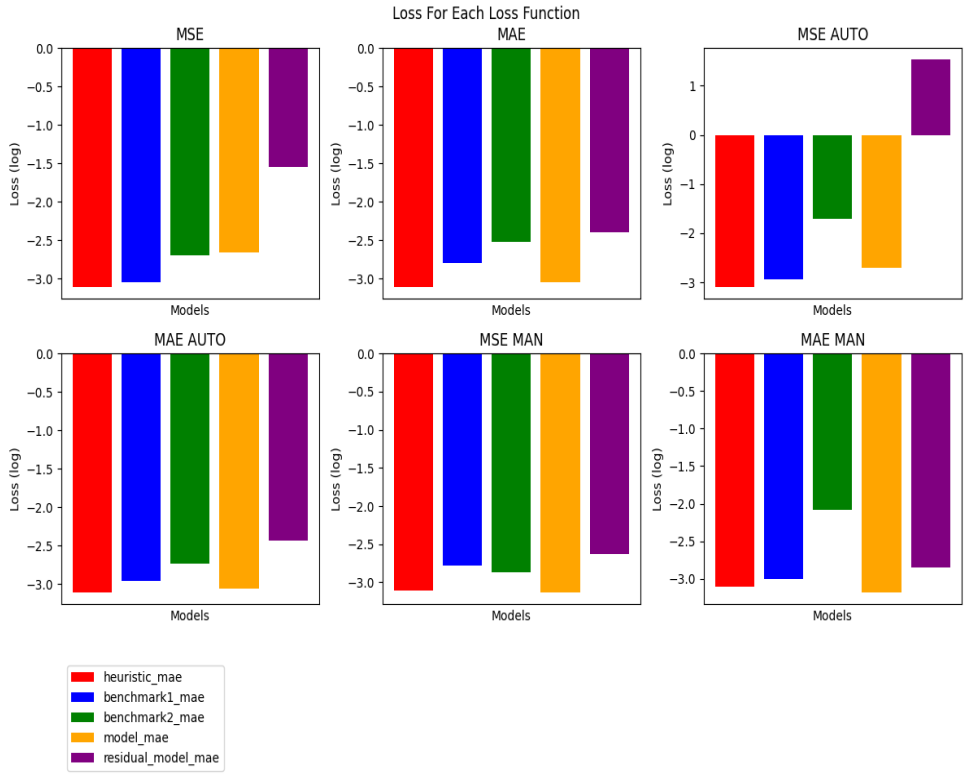


FIGURE 5.4: The loss for each defined loss function over the individually normalized 292 DEM dataset

## 5.2 Model Complexity and Hyperparameter tuning

### Predicting Water Depth

Two methods were evaluated. Models predicting water depth directly and models predicting the difference in water depth. Fig 5.5 shows a box plot of loss for predicting water depth versus predicting the difference in water depth between time steps. It is done across different learning rates, model complexities and epoch number (2 and 10 epochs).

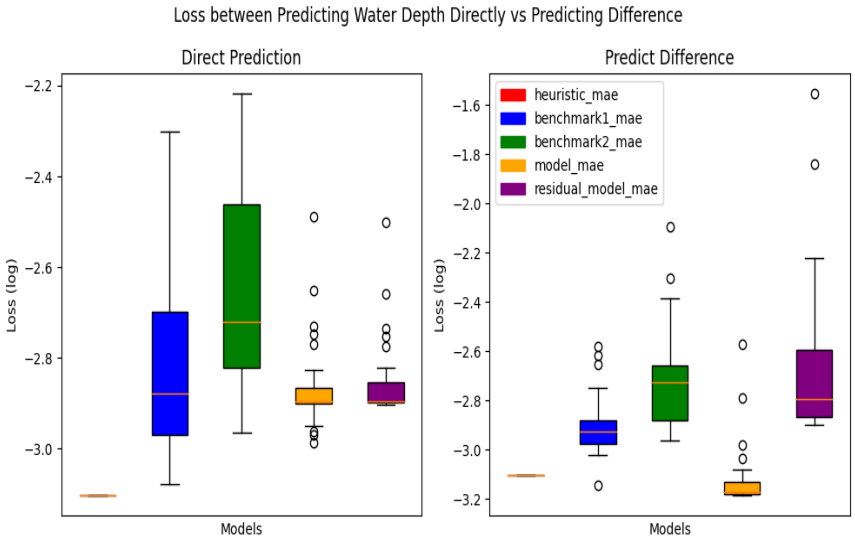


FIGURE 5.5: Loss for predicting water depth directly vs predicting the difference in water depth

Fig 5.5 shows that models trained to predict water depth directly have much more variability. Loss for predicting a single time step has more outliers and performs worse on average than predicting the difference in water depth. However, it seems that recurrent time step prediction performs better. Only models that predict difference in water depth over a single time step perform better than the heuristic. Based on these results, further evaluation is only based on predicting difference in water depth.

### Model Complexity, Predicting Difference in Water Depth

Fig. 5.6 shows the loss of different model complexities across epochs and learning rates. The shallow DepthwiseCNN model seems to show the best results. All model complexities out-perform the heuristic. Unfortunately, whilst generating data and deciding which models to do further testing on, only the best models were considered. In this case, the best performing model was a medium complexity model (see 5th row of Table 5.12). So further work was done on the medium complexity model. Further testing should be done on a simpler model [2], especially where the recurrent (and most useful) loss is concerned.

[2] see section 5.7 for a small further look into a simpler model and it the benefits it showed

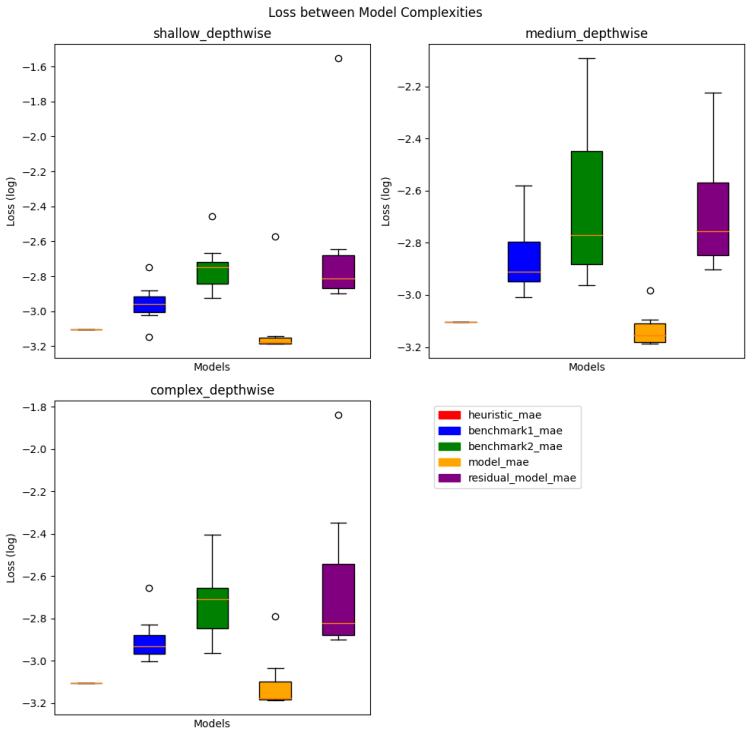


FIGURE 5.6: Loss for different model complexities that predict water difference

### Epochs and Learning Rate for Medium Complexity DepthwiseCNN

Fig 5.8 shows the model loss for the medium complexity DepthwiseCNN. This is across a variety of learning rates, which are shown in Table 5.2. There was very little difference between the epochs. A reason for this might be due to the stop loss callback option used during training. Overall, across all testing, models struggled to learn anything. A reason for this may be due to being trapped in local minima from the start. If the model was predicting no change, the loss was very low and even outperformed the heuristic. Fig 5.7 shows an example loss across all models (benchmark and DepthwiseCNN model). Where m is the model, b1, is benchmark 1 and b2 is benchmark 2. The reason for the large difference in loss is due to the type of loss functions employed. The benchmark models were always trained using standard MAE and default learning rate of 1e-3. The custom loss functions used for training the model of interest always increased the loss by 1000 times. The motivation for this was it would be easier to for the model to learn from a larger loss than a tiny loss.

From Table 5.2, the different learning rates had little effect on model performance. All learning rates performed better than the heuristic for medium complexity models. However, there did seem to be an effect on recurrent time step predictions. A learning rate of 1e-4 seemed to show the most promise, at least for recurrent time step predictions so this is what we use moving forward, unless otherwise specified.

### Optimal Weights for MAE

Table 5.3 shows the loss for different weights for the Manual MAE loss function. (see Fig. 5.9 for a visual representation).

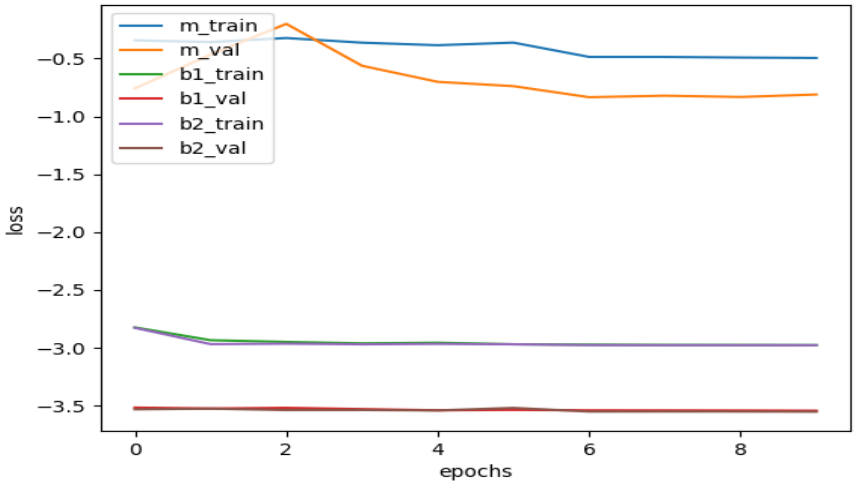


FIGURE 5.7: Example training loss for medium depthwise model and benchmarks

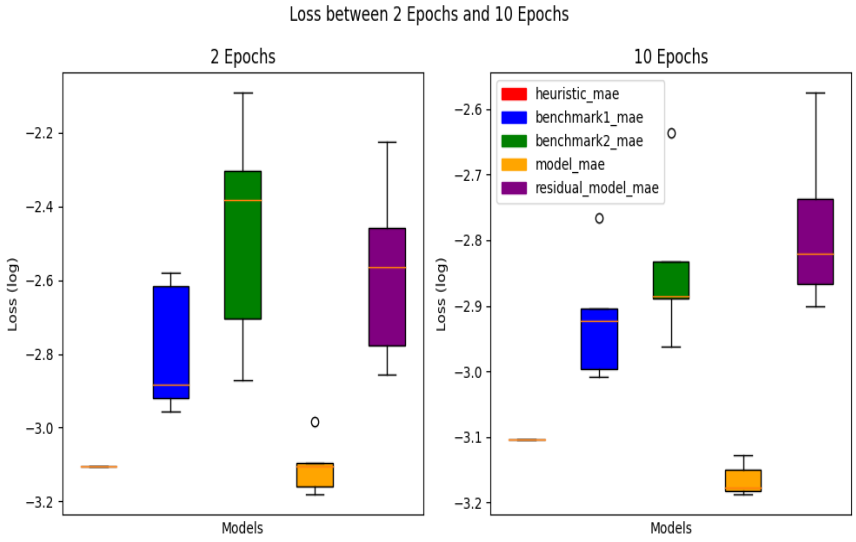


FIGURE 5.8: Loss for medium complexity DepthwiseCNN model over two different epochs

TABLE 5.2: Final losses for different learning rate with Medium Complexity

| Name | H | BM1 | BM2 | Model | Recurrent |
|---|---|---|---|---|---|
| 1e-3 | 0.000787 | 0.001194 | 0.001092 | 0.000665 | 0.001510 |
| 2e-3 | 0.000787 | 0.001010 | 0.001302 | 0.000658 | 0.001360 |
| 1e-4 | 0.000787 | 0.001714 | 0.002307 | 0.000648 | 0.001254 |
| 1e-2 | 0.000787 | 0.001246 | 0.001473 | 0.000744 | 0.002659 |
| 5e-4 | 0.000787 | 0.000982 | 0.001290 | 0.000707 | 0.001836 |

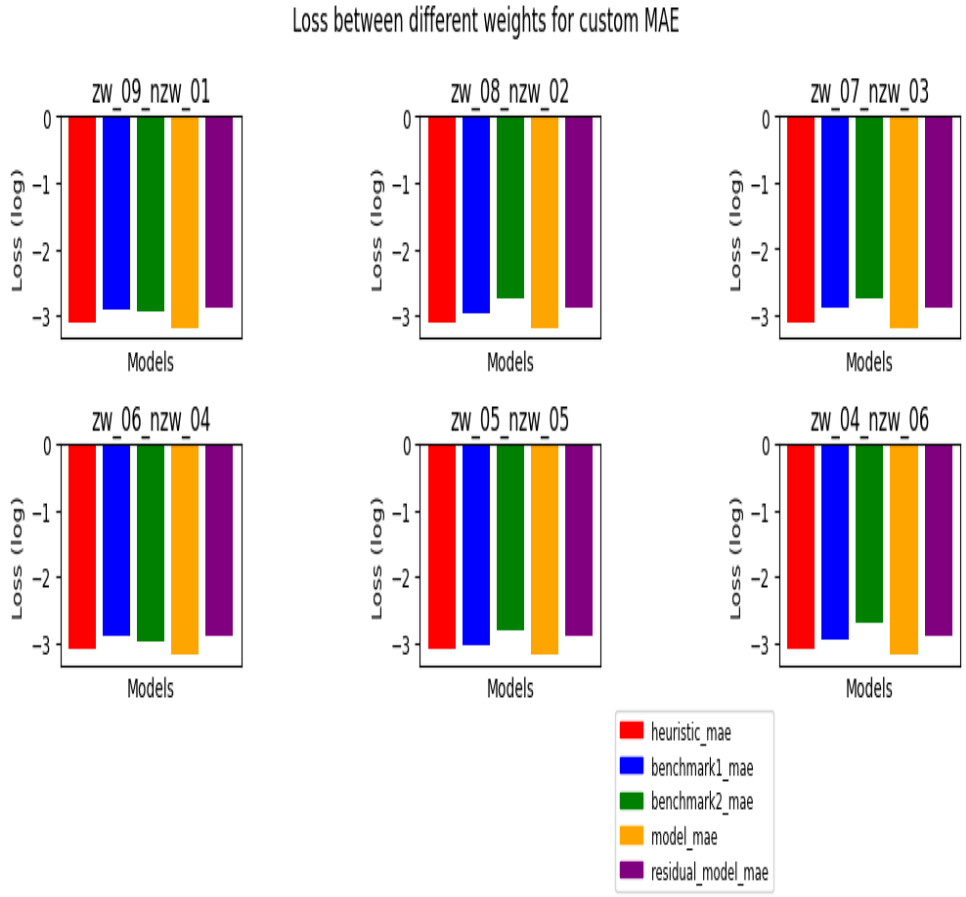


FIGURE 5.9: Loss for Different Weights of MAE

TABLE 5.3: The loss for different weightings in the MAE manuel loss function

| Name | H | BM1 | BM2 | Model | Recurrent |
|---|---|---|---|---|---|
| 09 \| 01 | 0.000787 | 0.001204 | 0.001137 | 0.000655 | 0.001322 |
| 08 \| 02 | 0.000787 | 0.001042 | 0.001830 | 0.000654 | 0.001335 |
| 07 \| 03 | 0.000787 | 0.001276 | 0.001845 | 0.000649 | 0.001267 |
| 06 \| 04 | 0.000787 | 0.001248 | 0.001051 | 0.000652 | 0.001302 |
| 05 \| 05 | 0.000787 | 0.000894 | 0.001517 | 0.000649 | 0.001260 |
| 04 \| 06 | 0.000787 | 0.001127 | 0.002029 | 0.000649 | 0.001263 |

Where the name indicates zero weight and non zero weight respectively. (eg. zero weight of 0.9 and non zero weight of 0.1, i.e. 0.9 | 0.1).

Since the Manual MAE loss function performed the best. The correct weight ratio was searched for. In Table 5.3, we see no meaningful difference (at most 0.000006 difference between weights.) This is a surprising result but as can be seen in Fig 5.7 models struggle to learn in general. Manual MAE with weighting of 0.5 | 0.5 was used for the rest of the evaluations unless otherwise specified.

#### Shuffling the Data

] Table 5.4 shows the results of shuffling the data using the function 'train _ test _ split()' from the python library sklearn with kwarg** shuffle=True. There was no meaningful difference between shuffling the data or not.

TABLE 5.4: The loss for shuffled or not shuffled

| Name | H | BM1 | BM2 | Model | Recurrent |
|---|---|---|---|---|---|
| Shuffled | 0.000787 | 0.002165 | 0.001247 | 0.000648 | 0.001255 |
| Sequential | 0.000787 | 0.001578 | 0.001307 | 0.000649 | 0.001263 |

## 5.3 Additional Features

Fig 5.10 shows the loss of models between the different feature datasets. From this plot it is clear that the original, independently normalized dataset shows the most constant and lowest losses across the board. So this is the dataset that will be considered for further evaluation. The motivation added rainfall directly to the water depth, is this is how the CADDIE model works. Unfortunately, due to the fact there are so little features anyway, the model was not able to correctly learn with this data. Although the model still outperformed the heuristic. Then the mask dataset is tested. The motivation is again because of the CADDIE model. Areas outside the catchment are considered run-off and some masked areas might make it into the training dataset. It would be hard for the model to differentiate the zero padding (which is there to preserve the shape of the input) from a masked area that should be run-off.

TABLE 5.5: Six training configurations testing two different weights for MAE Manuel and three lr across the original, normalized dataset

| Name | H | BM1 | BM2 | Model | Recurrent |
|---|---|---|---|---|---|
| w11 | 0.000787 | 0.001163 | 0.001336 | 0.000669 | 0.001453 |
| w12 | 0.000787 | 0.000958 | 0.001636 | 0.000726 | 0.002032 |
| w13 | 0.000787 | 0.001284 | 0.001447 | 0.000658 | 0.001350 |
| w21 | 0.000787 | 0.001067 | 0.002772 | 0.001642 | 0.014677 |
| w22 | 0.000787 | 0.000942 | 0.002398 | 0.000657 | 0.001403 |
| w23 | 0.000787 | 0.000990 | 0.002026 | 0.000648 | 0.001255 |

Where the first number indicates the weight ratio,

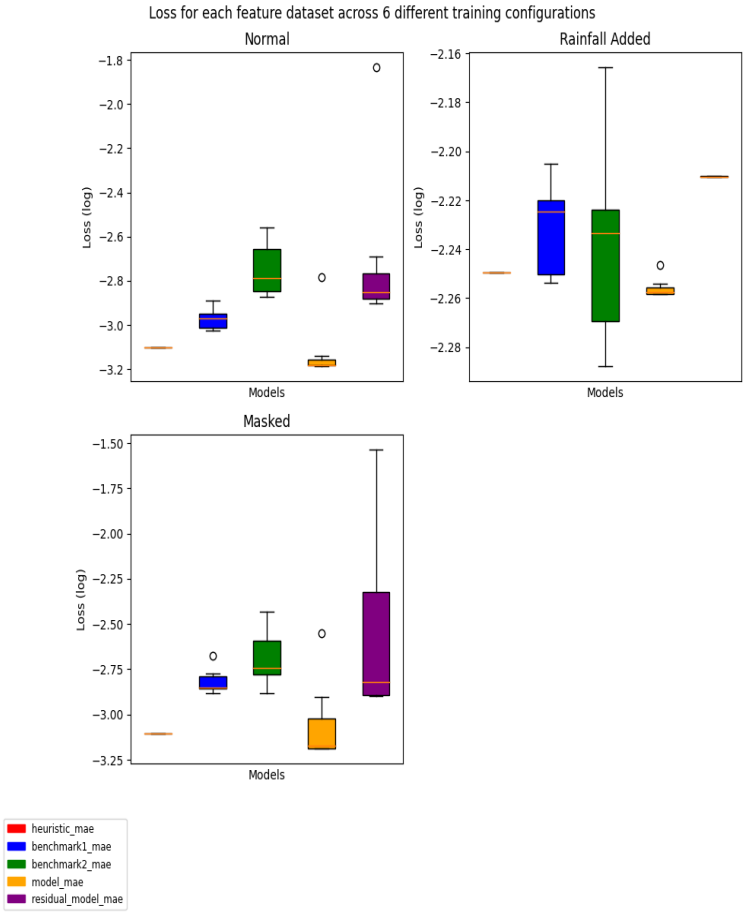


FIGURE 5.10: Loss for Three different Datasets with Different Features across varying training configurations

1. 0.5 | 0.5
2. 0.2 | 0.8

And the second number indicates the learning rate,

1. 1e-2
2. 1e-3
3. 1e-4

For example [w11] from Table 5.5 is the weight ratio of 0.5 zero weight and 0.5 non zero weight, with a learning rate of 1e-2. Here we see that such a high learning rate negatively impacts models and reaffirms that the weights had little impact except in the case of [w21] which performed much worst than the rest in terms of loss. One "success" was for a different weighting scheme but was not able to be reproduced can be seen in Fig 5.11. The it was trained on the original, independently normalized dataset, with a learning rate of 1e-4, using the Manual MAE loss function with weights 0.1 | 0.9.

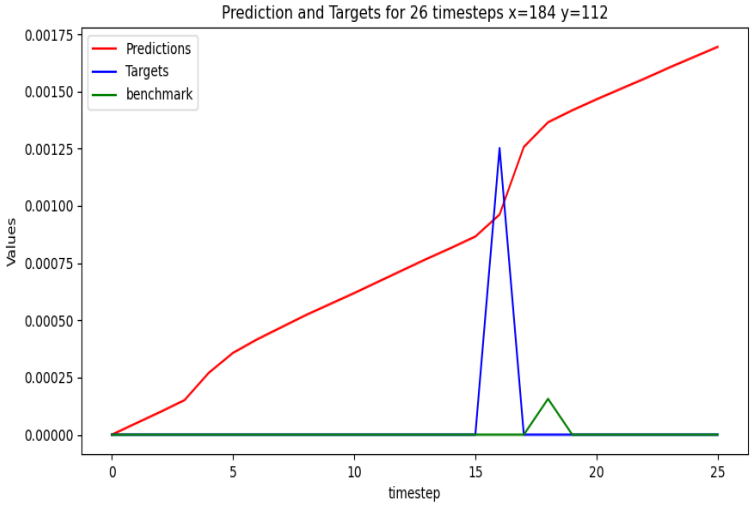


FIGURE 5.11: Medium depthwise model random cell residual time step predictions

This figure shows the ability of the model to learn a non-linear recurrent prediction, which was not seen prior, and provides some glimmer of hope that a DL model is capable of learning flood modeling.

## 5.4 Inception Inspired Model

In this section, column names are w1, w2, w3, They represent the following training regime,

w1 20 Epochs, learning rate of 1e-2, Manual MAE with zero weight 0,2 and non zero weight of 0,8

w2 20 Epochs, learning rate of 1e-3, Manual MAE with zero weight 0,2 and non zero weight of 0,8

w3 20 Epochs, learning rate of 1e-3, Manual MAE with zero weight 0,1 and non zero weight of 0,9

The motivation for using this model is it's ability to generate it's own feature maps using a variety techniques and then concatenating the results, which is then fed into into another sequential layer. Overall the results are quite similar to the DepthwiseCNN models but did perform, albeit, slightly better.

### Training Across Different Feature Datasets

Fig. 5.12, 5.13, 5.14 are the plots showing the losses of two different model complexities, simple and medium, across varying learning rates and weights for the Manual MAE (see above). The models performed the best on the independently normalized dataset, as with the DepthwiseCNN model from the previous section. However, it seems that the masked dataset showed less variability for recurrent predictions. Fig. 5.15 shows that the simple complexity Inception Inspired model also manages to learn something for recurrent predictions.

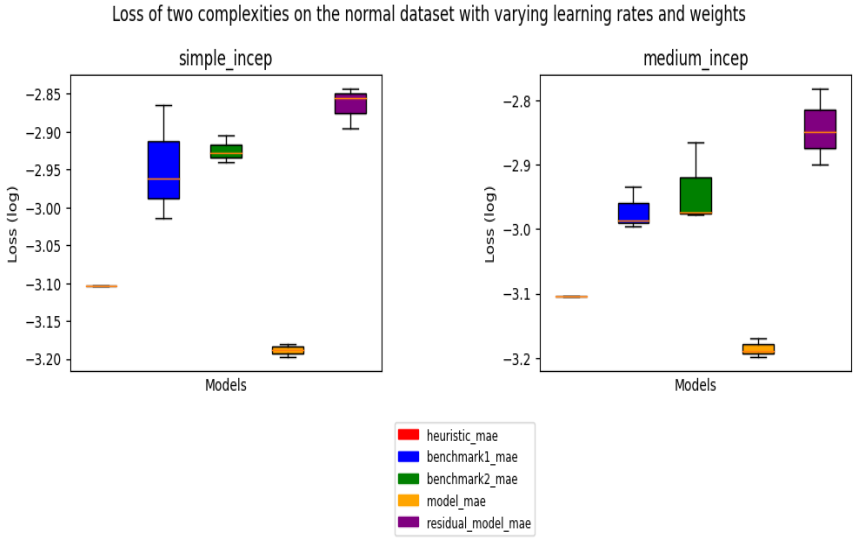


FIGURE 5.12: Loss for Inception Inspired model for two different complexities on the normal dataset

### Simple model

Table 5.6 shows that the different training regimes don't have meaningful impact on results. An example loss can be seen in Fig 5.16, where the results from training are very reminiscent of the DepthwiseCNN architecture.

In Table 5.7 we see that the model has the worst overall loss when comparing against the other feature datasets.

### Medium Model

The Medium complexity model increases parameter count dramatically. The results are shown below.

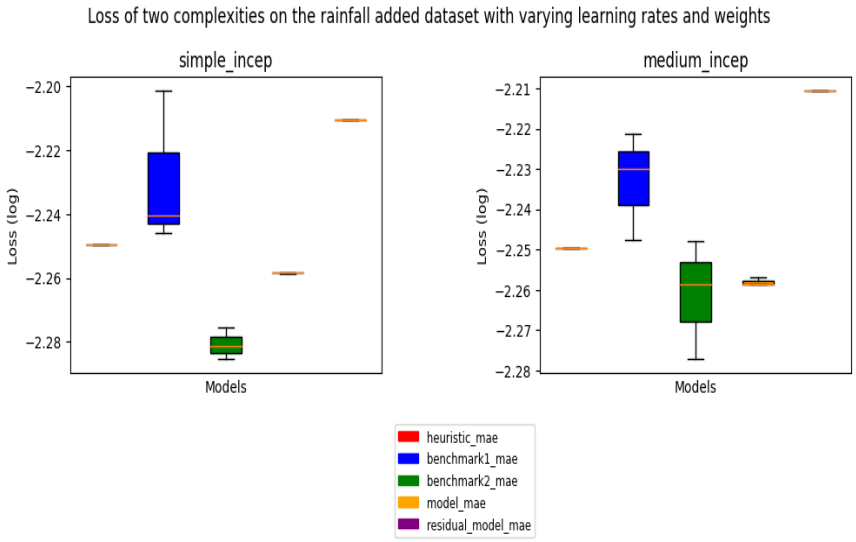


FIGURE 5.13: Loss for Inception Inspired model for two different complexities on the added rainfall dataset

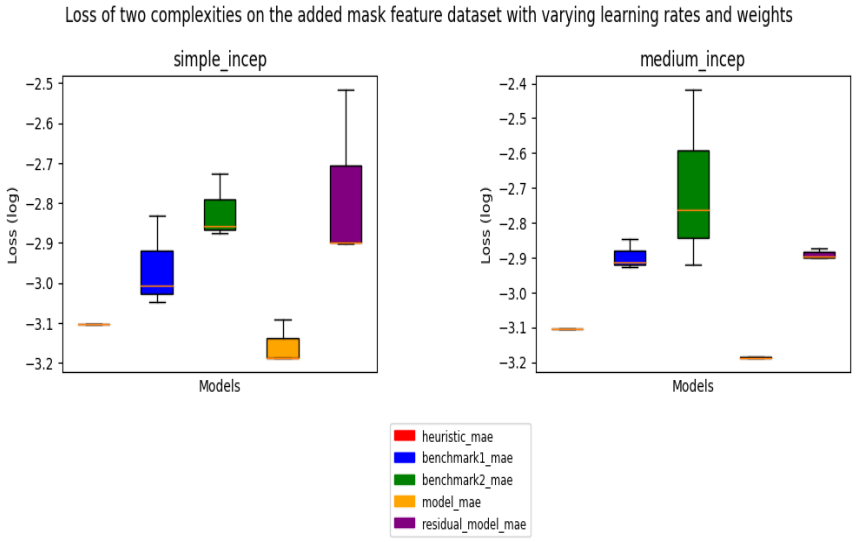


FIGURE 5.14: Loss for Inception Inspired model for two different complexities on the added masked dataset

TABLE 5.6: Three training configurations on the normal dataset for simple inception based model

| Name | H | BM1 | BM2 | Model | Recurrent |
|---|---|---|---|---|---|
| w1 | 0.000787 | 0.000966 | 0.001179 | 0.000661 | 0.001437 |
| w2 | 0.000787 | 0.001093 | 0.001247 | 0.000648 | 0.001272 |
| w3 | 0.000787 | 0.001366 | 0.001146 | 0.000634 | 0.001396 |

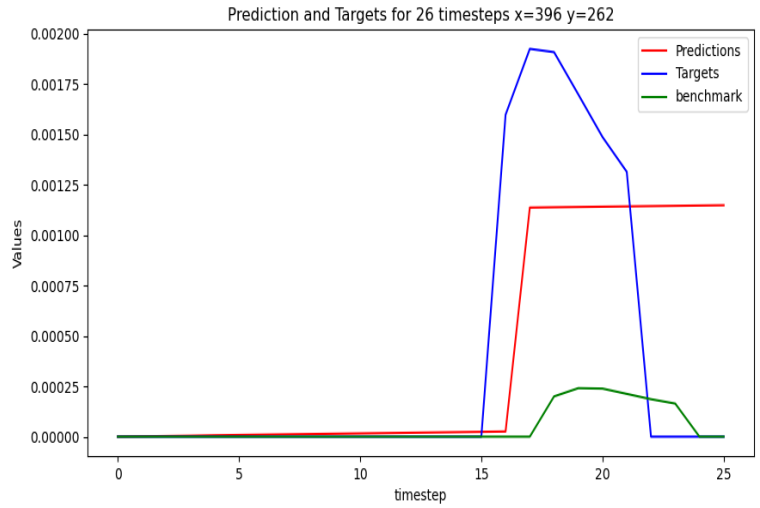


FIGURE 5.15: Simple Inception Inspired model on original independently normed dataset for a random cell predicting recurrently

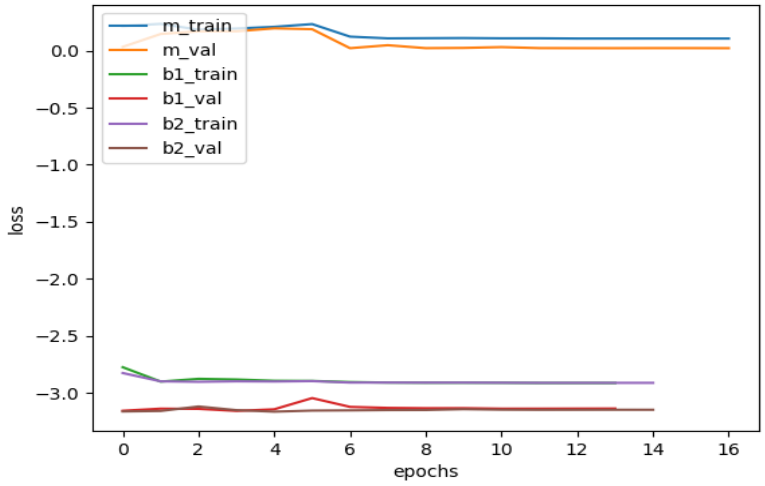


FIGURE 5.16: Loss during training of simple inception inspired model versus benchmark models

TABLE 5.7: Three training configurations on the rainfall added dataset for simple inception based model

| Name | H | BM1 | BM2 | Model | Recurrent |
|---|---|---|---|---|---|
| w1 | 0.005629 | 0.005676 | 0.005305 | 0.005517 | 0.006159 |
| w2 | 0.005629 | 0.006293 | 0.005231 | 0.005514 | 0.006159 |
| w3 | 0.005629 | 0.005750 | 0.005183 | 0.005519 | 0.006159 |

TABLE 5.8: Three training configurations on the mask added dataset for simple inception based model

| Name | H | BM1 | BM2 | Model | Recurrent |
|---|---|---|---|---|---|
| w1 | 0.000787 | 0.000986 | 0.001333 | 0.000811 | 0.003052 |
| w2 | 0.000787 | 0.001469 | 0.001390 | 0.000648 | 0.001262 |
| w3 | 0.000787 | 0.000897 | 0.001878 | 0.000648 | 0.001253 |

TABLE 5.9: Three training configurations on the normal dataset for medium inception based model

| Name | H | BM1 | BM2 | Model | Recurrent |
|---|---|---|---|---|---|
| w1 | 0.000787 | 0.001032 | 0.001054 | 0.000676 | 0.001655 |
| w2 | 0.000787 | 0.001166 | 0.001064 | 0.000632 | 0.001418 |
| w3 | 0.000787 | 0.001011 | 0.001362 | 0.000648 | 0.001258 |

TABLE 5.10: Three training configurations on the rainfall added dataset for medium inception based model

| Name | H | BM1 | BM2 | Model | Recurrent |
|---|---|---|---|---|---|
| w1 | 0.005629 | 0.005655 | 0.005513 | 0.005534 | 0.006159 |
| w2 | 0.005629 | 0.006006 | 0.005282 | 0.005514 | 0.006160 |
| w3 | 0.005629 | 0.005888 | 0.005650 | 0.005514 | 0.006160 |

TABLE 5.11: Three training configurations on the mask added dataset for medium inception based model

| Name | H | BM1 | BM2 | Model | Recurrent |
|---|---|---|---|---|---|
| w1 | 0.000787 | 0.001427 | 0.003831 | 0.000654 | 0.001342 |
| w2 | 0.000787 | 0.001183 | 0.001719 | 0.000647 | 0.001254 |
| w3 | 0.000787 | 0.001222 | 0.001197 | 0.000650 | 0.001267 |

From Tables 5.9, 5.10, 5.11, The results for the masked and normal datasets are similar but best results are still seen in the original independent dataset. Although the medium complexity Inception Inspired model outperforms the simple version, the increased parameter count and time needed to train also needs to be considered in future evaluations. In the below section, results across all training of all models are compared.

## 5.5 Final Results

TABLE 5.12: The Top Five Models Across all Training SS

| Name | H | BM1 | BM2 | Model | Recurrent |
|---|---|---|---|---|---|
| 1 | 0.000787 | 0.001166 | 0.001064 | 0.000632 | 0.001418 |
| 2 | 0.000787 | 0.001366 | 0.001146 | 0.000634 | 0.001396 |
| 3 | 0.000787 | 0.001183 | 0.001719 | 0.000647 | 0.001254 |
| 4 | 0.000787 | 0.001011 | 0.001362 | 0.000648 | 0.001258 |
| 5 | 0.000787 | 0.002165 | 0.001247 | 0.000648 | 0.001255 |

Table 5.12 shows the top performing models for predicting a single time step (SS) ahead, overall across all testing. The Name column has the following keys,

1. Medium Complexity Inception Inspired Model on the normalized dataset with the original three features. Trained with 20 Epochs, a learning rate of 1e-3, Manual MAE loss with zero weighting of 0,2 and non zero weighting 0,8.

2. Simple Complexity Inception Inspired Model on the normalized dataset. Trained with 20 Epochs, a learning rate of 1e-3, Manual MAE loss with zero weighting of 0,1 and non zero weighting 0,9.

3. Medium Complexity Inception Inspired Model on the normalized dataset with an additional mask feature. Trained with 20 Epochs, a learning rate of 1e-3, Manual MAE loss with zero weighting of 0,2 and non zero weighting 0,8.

4. Medium Complexity Inception Inspired Model on the normalized dataset with the original three features. Trained with 20 Epochs, a learning rate of 1e-3, Manual MAE loss with zero weighting of 0,1 and non zero weighting 0,9.

5. Medium complexity DepthwiseCNN model on the normalized dataset with original features. Trained with 8 Epochs,a learning rate of 1e-4, Manual MAE loss with zero weighting of 0,5 and non zero weighting of 0,5.

Table 5.13 shows the top five models based on recurrent water depth prediction across all training. The name column has the following key-value pairs,

1. Medium Complexity DepthwiseCNN model predicting water depth directly. Trained on the normalized dataset with 10 epochs, a learning rate of 1e-4.

2. Simple Complexity Inception Inspired model the normalized dataset with the added mask feature. Trained with 20 Epochs, a learning rate of 1e-3, Manual MAE loss with zero weighting of 0,1 and non zero weighting 0,9.

TABLE 5.13: The Top Five Models Across all Training For Recurrent Steps

| Name | H | BM1 | BM2 | Model | Recurrent |
|---|---|---|---|---|---|
| 1 | 0.000787 | 0.001502 | 0.001632 | 0.001253 | 0.001253 |
| 2 | 0.000787 | 0.000897 | 0.001878 | 0.000648 | 0.001253 |
| 3 | 0.000787 | 0.001183 | 0.001719 | 0.000647 | 0.001254 |
| 4 | 0.000787 | 0.002816 | 0.003378 | 0.001254 | 0.001254 |
| 5 | 0.000787 | 0.001714 | 0.002307 | 0.000648 | 0.001254 |

3. Medium Complexity Inception Inspired Model on the normalized dataset with the added mask feature, predicting difference in water depth. Trained with 20 Epochs, a learning rate of 1e-3, Manual MAE loss with zero weighting of 0,2 and non zero weighting 0,8.

4. Medium Complexity DepthwiseCNN model predicting water depth directly. Trained on the normalized dataset with 2 epochs, a learning rate of 1e-4.

5. Medium Complexity DepthwiseCNN model predicting the difference in water depth. Trained on the normalized dataset with 10 epochs, a learning rate of 1e-4.

TABLE 5.14: The Worst Five Models Across all Training For SS

| Name | H | BM1 | BM2 | Model | Recurrent |
|---|---|---|---|---|---|
| 1 | 0.002279 | 81.628900 | 106.629910 | 44.619150 | 648.459210 |
| 2 | 0.002279 | 78.769480 | 111.990715 | 32.265385 | 379.157455 |
| 3 | 0.002551 | 14.399827 | 25.591928 | 16.135822 | 187.824300 |
| 4 | 0.002551 | 21.657652 | 24.267190 | 8.066945 | 101.351124 |
| 5 | 0.002551 | 18.857510 | 27.992000 | 5.856058 | 83.558502 |

Table 5.14 shows the worst five performing models for predicting a single time step (SS) ahead, overall across all testing. The Name column has the following keys,

1. Inital Depthwise Model using classic MSE on the DEM 26 dataset where everything was min max normalized using the DEM. Trained using 2 Epochs, default learning rate of 1e-3.

2. Inital Depthwise Model using Manual MSE on the DEM 26 dataset where everything was min max normalized using the DEM. Trained using 2 Epochs, default learning rate of 1e-3. zero weight of 0.8 and non zero weight of 0.2.

3. Inital Depthwise Model using classic MSE on the DEM 819 dataset where everything was min max normalized using the DEM. Trained using 2 Epochs, default learning rate of 1e-3.

4. Inital Depthwise Model using Manual MSE on the DEM 819 dataset where everything was min max normalized using the DEM. Trained using 2 Epochs, default learning rate of 1e-3. zero weight of 0.8 and non zero weight of 0.2.

5. Inital Depthwise Model using auto MSE on the DEM 819 dataset where everything was min max normalized using the DEM. Trained using 2 Epochs, default learning rate of 1e-3.

TABLE 5.15: The Worst Five Models Across all Training For Recurrent Steps

| Name | H | BM1 | BM2 | Model | Recurrent |
|---|---|---|---|---|---|
| 1 | 0.002279 | 81.628900 | 106.629910 | 44.619150 | 648.459210 |
| 2 | 0.002279 | 78.769480 | 111.990715 | 32.265385 | 379.157455 |
| 3 | 0.002551 | 14.399827 | 25.591928 | 16.135822 | 187.824300 |
| 4 | 0.002440 | 4.586013 | 38.804940 | 4.971159 | 164.466783 |
| 5 | 0.001346 | 0.001753 | 0.002407 | 0.002654 | 119.582068 |

Table 5.15 shows the worst five performing models for predicting recurrent time steps ahead, overall across all testing. The Name column has the following keys,

1. Inital Depthwise Model using classic MSE on the DEM 26 dataset where everything was min max normalized using the DEM. Trained using 2 Epochs, default learning rate of 1e-3.

2. Inital Depthwise Model using Manual MSE on the DEM 26 dataset where everything was min max normalized using the DEM. Trained using 2 Epochs, default learning rate of 1e-3. zero weight of 0.8 and non zero weight of 0.2.

3. Inital Depthwise Model using classic MSE on the DEM 819 dataset where everything was min max normalized using the DEM. Trained using 2 Epochs, default learning rate of 1e-3.

4. Inital Depthwise Model using auto MAE on the DEM 292 dataset where everything was min max normalized using the DEM. Trained using 2 Epochs, default learning rate of 1e-3.

5. Inital Depthwise Model using auto MSE on the DEM 819 dataset where features are normalized independently. Trained using 2 Epochs, default learning rate of 1e-3.

The following plots where created after all initial testing to try and reproduce the models. Fig. 5.17 shows the water depths of the true values, heuristic model output, and deep learning model predictions of a random cell in the test set using a recurrent time step method prediction. The model used was the best performing model for a single time step prediction. (see Table. 5.13). Fig. 5.18 shows the same plot but predicting a single time step ahead. Fig. 5.19 and 5.20 are the plots respectively but using the best single time step model (see Table. 5.12).

Fig. 5.21 shows the spatial evolution of water depth on the test set and is used for comparison. Fig. 5.22 shows the best model from Table 5.12 and its spatial evolution of water depth over the test set. Fig. 5.23 shows the spatial evolution of water depth on the test set using recurrent predictions. All other models predicted 0. This model is a slightly adjusted model, trained on lower epochs (2) and adjusting MAE weighting to 0.5 | 0.5. Fig. 5.24 shows the recurrent preditions of a collection of random cells across the test set.

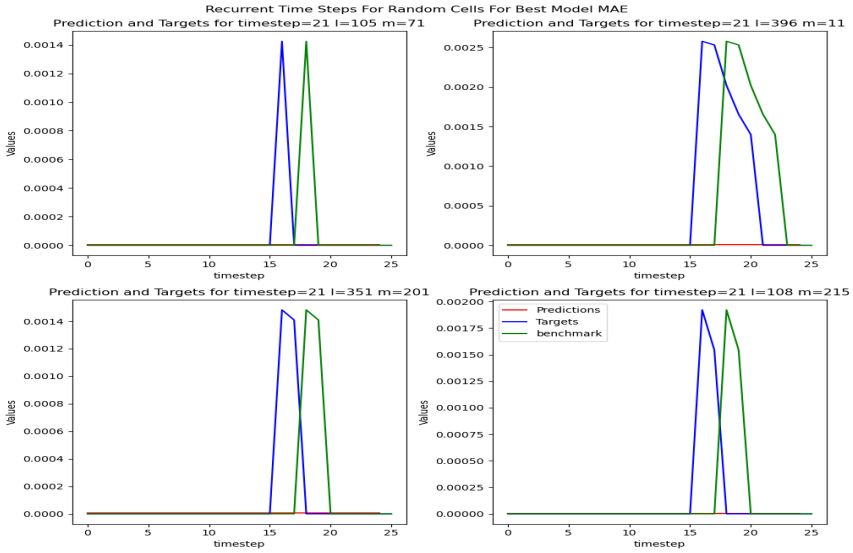


FIGURE 5.17: Water Depth Predictions recurrently over the test set timesteps

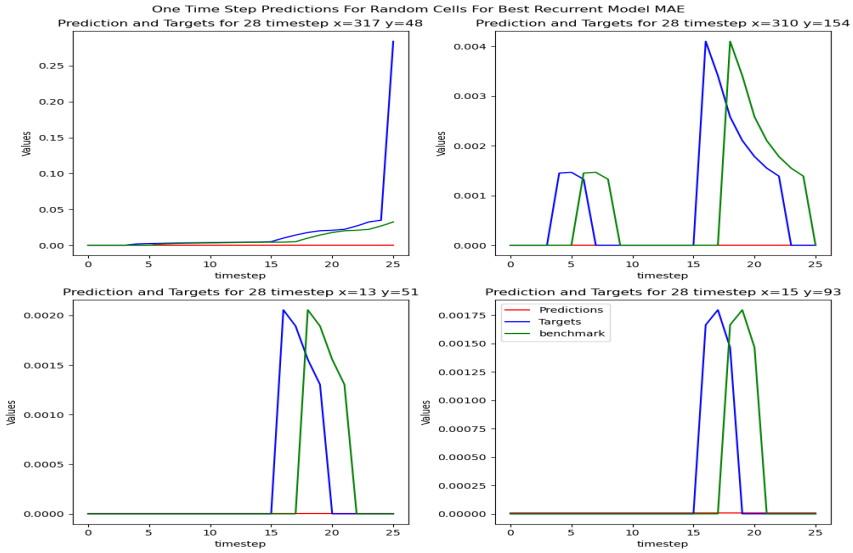


FIGURE 5.18: Best Recurrent Loss on single time step predictions over test set

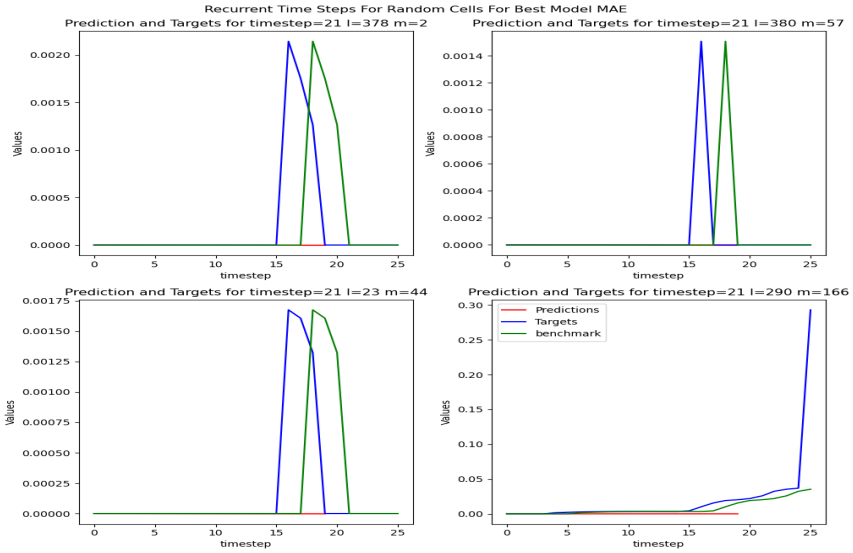


FIGURE 5.19: Best single step model on test set predicting water depth for recurrent time steps

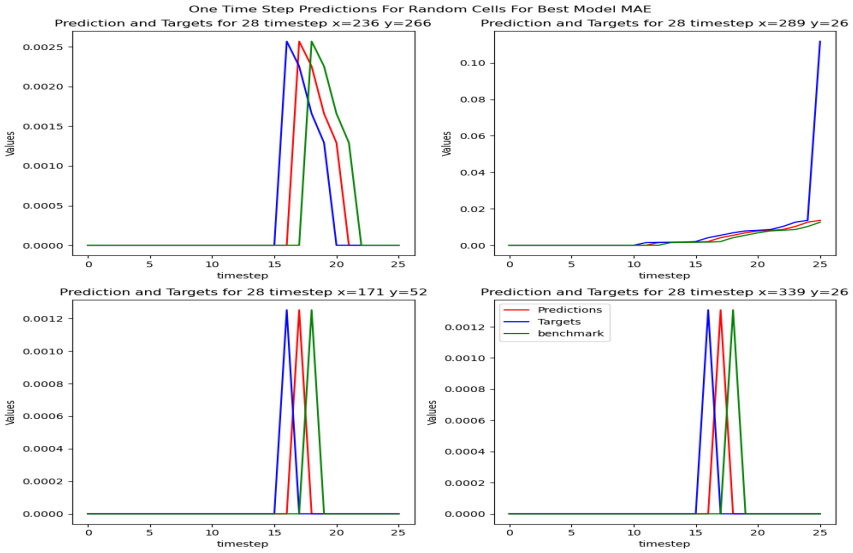


FIGURE 5.20: Best single time step model on predicting water depth for a single time step

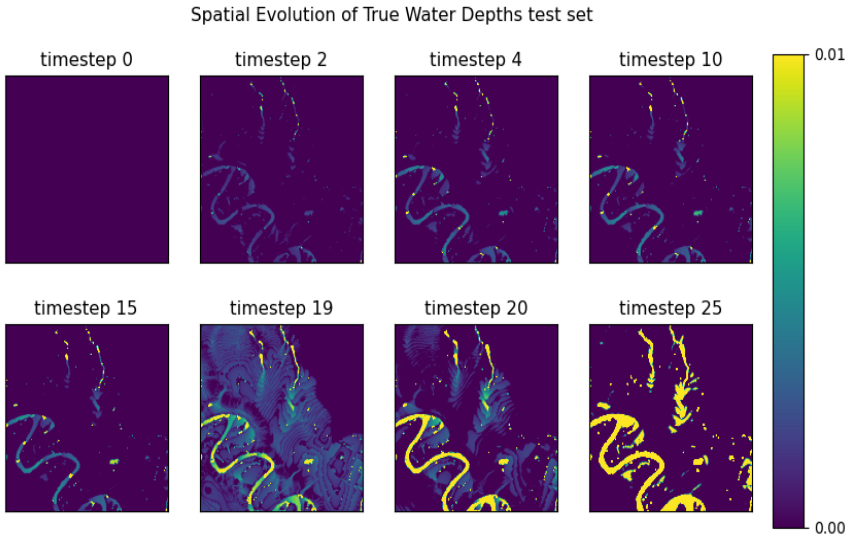


FIGURE 5.21: Spatial Evolution of True Water Depth for test set

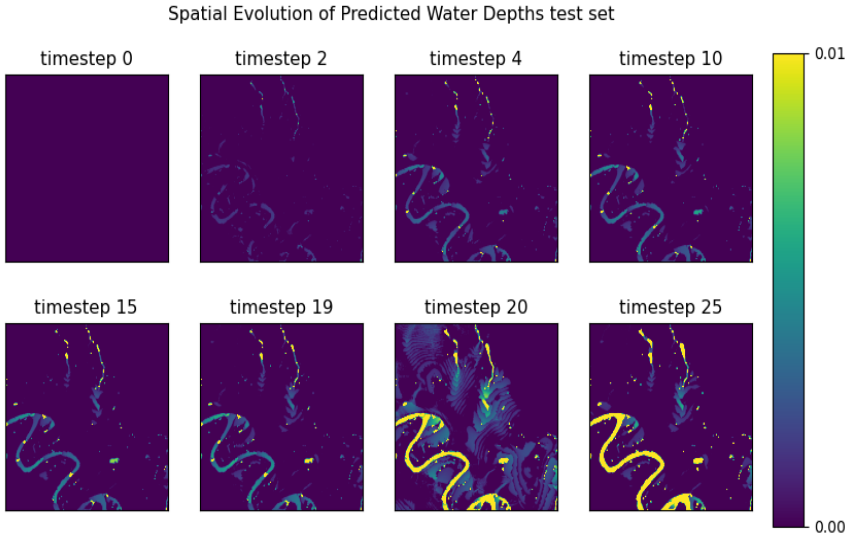


FIGURE 5.22: Spatial Evolution for the Best Model Predicting single time steps

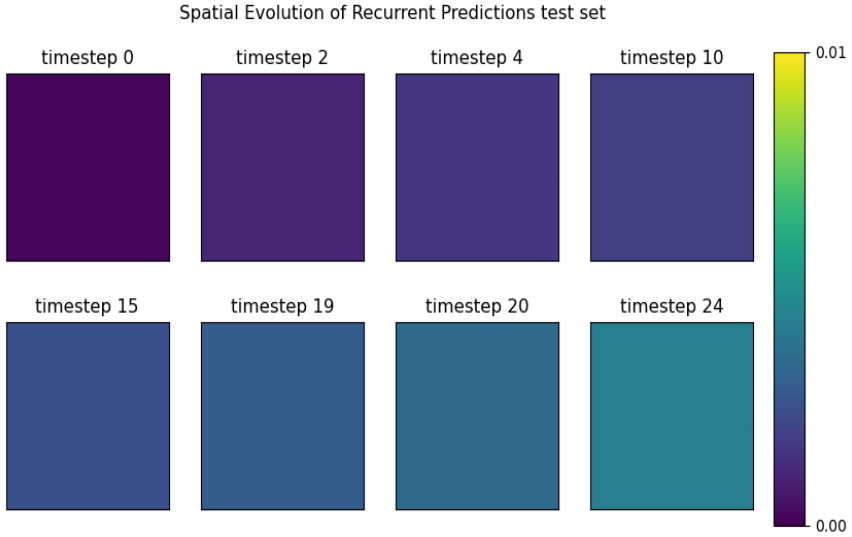


FIGURE 5.23: Spatial Evolution for the Best Model with adjusted weights (0.5, 0.5) Predicting Recurrent time steps

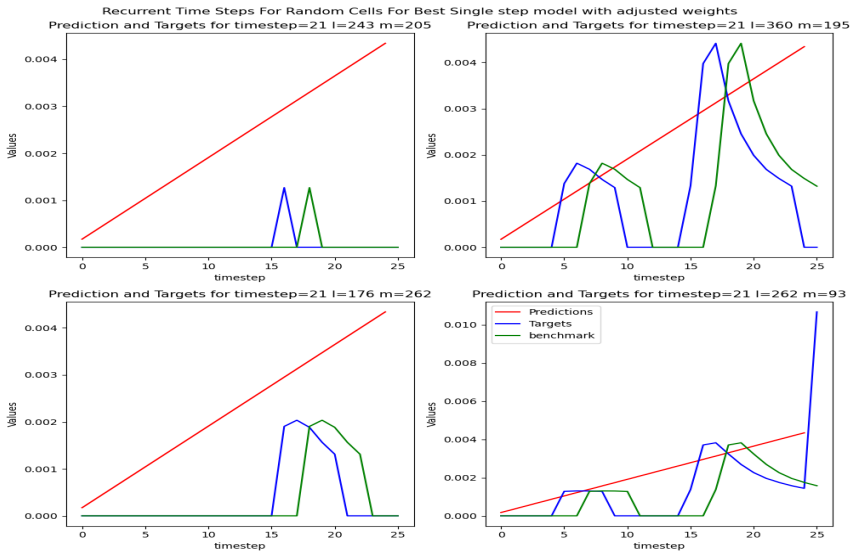


FIGURE 5.24: Spatial Evolution for the Best Model Predicting recurrent time steps but with adjusted weights (0.5, 0.5)

## 5.6 Limitations

Although the vast majority of models outperformed the benchmark models and heuristic, they were unable to recurrently predict water depths across time points. Even single time step prediction was only predicting no change in water depth despite trying to account for this in the loss functions. There are many reasons for this which are outlined here,

Normalization Strategy: Only three normalization techniques were tested. This was a very difficult problem to tackle due to the range within features. Perhaps the way normalization was done, hindered the models ability to learn

Computational Limitations: Training data had to be sub sampled. Since time steps were 10 minute intervals and the DEM has a spatial resolution of 1 m, water is able to propagate many cells downstream in a single time step. Outside of the local neighborhood. When sub sampling is used, the model does not have access to this information. Which may have resulted in the models' being unable to learn correctly. This sub sampling technique also makes padding more of an issue. How is the model meant to know whether the data is outside a catchment area or whether it is just zero padding? It may be unable to determine this information. Or if a sub sample of data is upstream or downstream of another subsample. All of this is potentially useful information that the model may not have had access to due to computational constraints.

Loss Function and Metric: Perhaps the metric of evaluation could be improved. MAE error across the whole dataset may prioritize the model to predict no change as many of the cells are 0 from the simulated data.

Unbalanced Data: The rainfall patterns are quite strange and most of the rainfall occurs in small time windows. Therefore most cells, at most time steps have 0 water depth. This means predicting 0 water depth results in a very low loss and the model easily recognizes this fact.

Single Time Step Prediction: A problem with this approach is not practically useful unless emergent properties like recurrent predictions are obtained which was not the case in this evaluation.

Evaluation: Metric had a fatal flaw and all testing done ultimately drove the model to predict no change in water depth. Even picking DEM 292 and the test set with 2 short spikes in rain fall (see DEM 292 in Fig. 4.4). Future work should consider each training outcome in a case by case manner with far more careful evaluation to what the model is doing. A small experiment was performed in Section 5.7 to highlight this.

## 5.7 Extras

This section provides results from further testing that was done after the initial write up. Some optimistic results are presented after discovering the evaluation methods above were driving the model to predict zero change in water depth. The model used for this evaluation is as follows:

```
    class SimpleCNN(tf.keras.Model):
        def __init__(self, model_config):
```

```
        super().__init__()
            self.config = model_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, 3]))

        def call(self, x):
            dx = self.dmodel(x)
            f1, f2, f3 = tf.unstack(x, axis=-1)
            f1 =tf.expand_dims(f1, -1)
            return dx + f1
```

With model summary,

```
Model: "simple_cnn_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
sequential_6 (Sequential)    (None, None, None, 1)     7489
=================================================================
Total params: 7,489
Trainable params: 7,489
Non-trainable params: 0
_________________________________________________________________
```

We see that very few parameters are contained in this model. In this case, only the DEM was normalized and the rest were left un modified. Using the Manual MAE, but this time with zero weighting of 0.1 and non zero weighting of 0.9, we find much more promising results. See figures below.

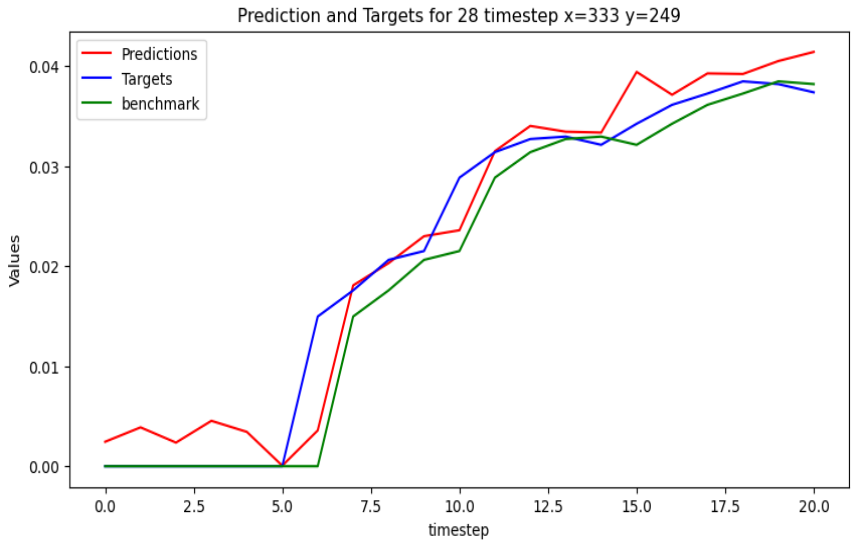


FIGURE 5.25: Single Step prediction versus heuristic and target for a random cell in the test dataset for DEM 26

It is very interesting to see that the recurrent predictions essentially are predicting

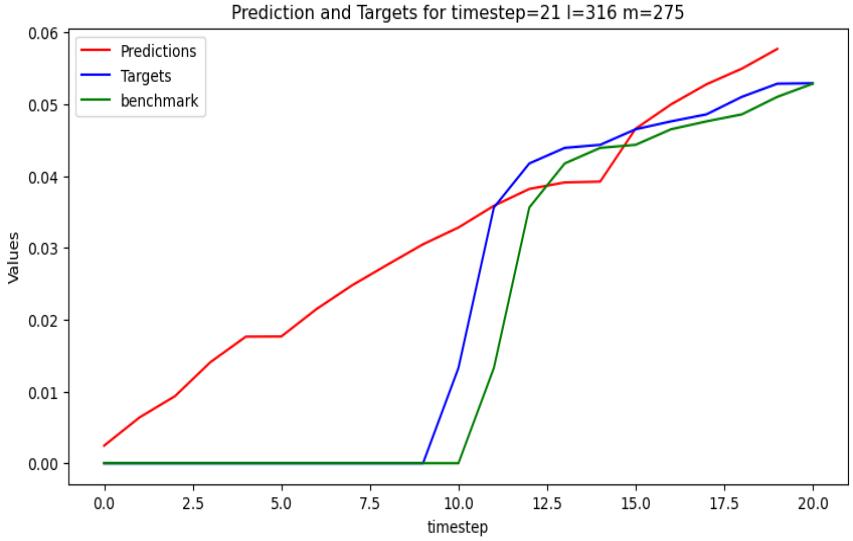


FIGURE 5.26: Recurrent time step prediction versus heuristic and target for a random cell in test dataset DEM 26

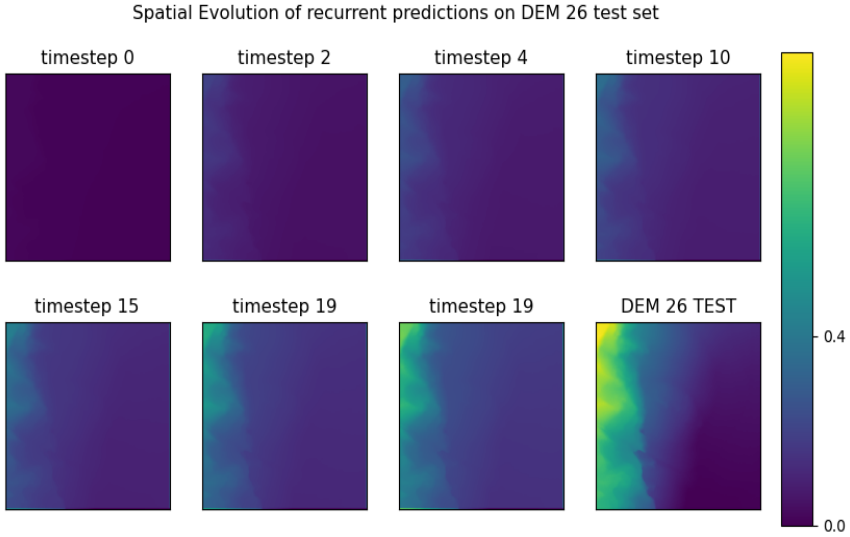


FIGURE 5.27: Spatial evolution for the recurrent time step predictions. The final image is the DEM for dataset 26

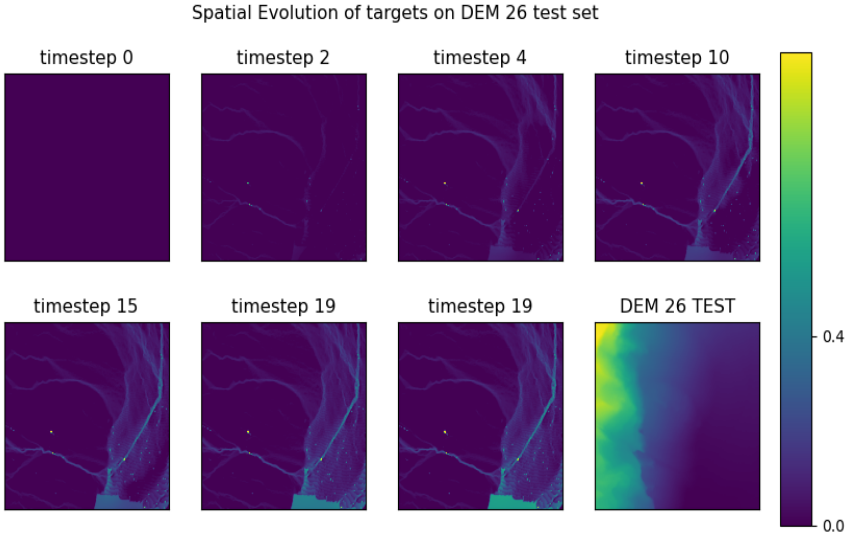


FIGURE 5.28: Spatial evolution for targets for test dataset DEM26. The final image is the DEM for dataset 26

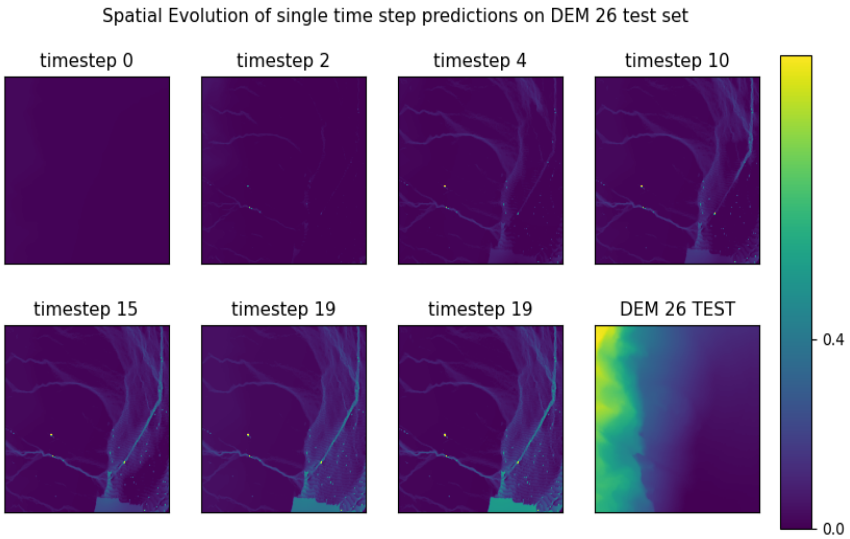


FIGURE 5.29: Spatial evolution for single time step predictions for test dataset DEM26. The final image is the DEM for dataset 26

the DEM map itself (see Fig 5.27), or at least its shape. This may be due to a bug of some kind but is interesting to see. Whereas the true values seem to be nothing like that (see Fig. 5.28). From Fig. 5.25 and 5.26 we see much more promising results than mentioned previously. These results were obtained using more training (50 epochs) and a learning rate of 0.001 with reduced learning rate as a call back. These results provide some hope that it is possible to model flooding behaviors with NCA. We can also see the similarities more visually between Fig 5.29 and 5.28. Time step 19 shows a slight difference in water depth, where the model predicted a higher water depth for a single cell. The emergent properties seen in the recurrent time step predictions are also interesting, since the model was not explicitly trained to predict recurrently! The results for recurrent predictions are poor, however there are positive signs that it is possible but further evaluation and testing will be required.

# Chapter 6

# Conclusion

## 6.1 Conclusion

To conclude, In relation to Objective (i), the vast majority of models trained managed to out performed the heuristic as well as the benchmark models, however, most of them predicted no change in water depth. Therefore, in relation to Objective (ii), single time step predictions did not perform as well as anticipated on the unseen rainfall event. After careful consideration of shortcomings and pitfalls, some further evaluation was done on a much simpler model (7000 parameters) and showed much more promising results in relation to Objective (i) and (ii). In relation to Objective (iii), emergent behavior was, in general, not observed, even after further evaluation. However, there were some glimmers of hope where models were able to predict non-linear increase and decrease to water depth based on rainfall, although strange behavior was also noted, where the model seemed to predict the exact shape of the DEM itself. Further evaluation is required to understand this strange behavior but we are optimistic that there is potential for the models to accurately predict water depth recurrently, even if it is not explicitly trained to do so!

## 6.2 Outlook and future work

Although the project was not a resounding success, some positives were seen in some of the results. The fact that some of the models were able to recurrently predict water depths in a non-linear fashion shows some promise for future evaluations. There are many factors which could have been improved in this thesis and there are many more deep learning techniques to try. Including using a training regime that was used by Mordvintsev et al. where back propagation is done through time. Computational limitations for training was a severe problem. Being forced to pad small subsections of the training data may have resulted in a loss of critical information. Also the evaluation itself was flawed and ultimately drove the models to predict no change in water depth. This was shown to be true when positive outcomes resulted from ignoring many of these metrics in a small re-evaluation (see Section 5.7) was done.

In future work, the use of High Performance Computing Cluster may be necessary. A lot more data could be simulated and run without worrying about padding within the catchment area. More work needs to be done to find a generalizable normalization technique as well as creating a more balanced dataset. Further works also needs to be done on finding a suitable loss function that drives the model to predict non

zero values, which may have been one of the leading cause of error in this thesis. Luckily these are relatively easily remedied. The way the results were obtained and shown, allows for an easy reroute in evaluation.

# Appendix A

# 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.

## 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 neighborhood 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; neighborhood (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. 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 neighbors 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 calculation 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 $\Delta t$ 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:

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

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. $\Delta WL_i$ 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

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

is applied to limit $\Delta V_i$, 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:

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

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:

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

Where $\theta$ 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 *A.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 $\theta$ are determined by numerical experiments and calibration.

### Time-Step Calculation

For most 2D hydraulic modeling, 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:

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

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

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

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

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

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 A.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:

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

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

# Appendix B

# Raw Output From Training

```
name,heuristic_mae,benchmark1_mae,benchmark2_mae,model_mae,residual_model_mae
initial_test_model_train_1_mse_un_normed_26,0.0022785888717288,0.17706834,0.065506816,0.0018053363,0.0131062043786263
initial_test_model_train_1_mae_un_normed_26,0.0022785888717288,0.034372505,0.19239059,0.0012089811,0.0108256550542878
initial_test_model_train_1_mse_auto_un_normed_26,0.0022785888717288,0.09283428,0.14614823,0.0016407528,0.0121898610401978
initial_test_model_train_1_mae_auto_un_normed_26,0.0022785888717288,0.04104698,0.073801585,0.0012650765,0.0114737847487118
initial_test_model_train_1_mse_weighted_un_normed_26,0.0022785888717288,0.049467638,0.10221555,0.0013220084,0.0110205996528985
initial_test_model_train_1_mae_weighted_un_normed_26,0.0022785888717288,0.094178386,0.046659555,0.0012358107,0.0111366017744339
initial_test_model_train_1_mse_un_normed_292,0.0024398706087699,0.06716451,0.022859398,0.0023319009,0.0167536369127991
initial_test_model_train_1_mae_un_normed_292,0.0024398706087699,0.0480997,0.027595503,0.0013804316,0.0066861019789543
initial_test_model_train_1_mse_auto_un_normed_292,0.0024398706087699,0.031091763,0.014013137,0.0017122391,0.0141857316170295
initial_test_model_train_1_mae_auto_un_normed_292,0.0024398706087699,0.045181967,0.014117654,0.001314322,0.0078721977852273
initial_test_model_train_1_mse_weighted_un_normed_292,0.0024398706087699,0.027381152,0.035884462,0.0013841033,0.007006884198725
initial_test_model_train_1_mae_weighted_un_normed_292,0.0024398706087699,0.038767345,0.031516783,0.001399385,0.0068956770554227
initial_test_model_train_1_mse_un_normed_709,0.0033332953612009,0.003979011,0.028122649,0.0031228988,0.0270364755959635
initial_test_model_train_1_mae_un_normed_709,0.0033332953612009,0.061910894,0.014540986,0.0018932822,0.0153214776347187
initial_test_model_train_1_mse_auto_un_normed_709,0.0033332953612009,0.023917101,0.033802394,0.002778097,0.0257706127570231
initial_test_model_train_1_mae_auto_un_normed_709,0.0033332953612009,0.080488764,0.02987295,0.0018895979,0.0144089194518597
initial_test_model_train_1_mse_weighted_un_normed_709,0.0033332953612009,0.046321664,0.01079995,0.0021118089,0.0164113486446882
initial_test_model_train_1_mae_weighted_un_normed_709,0.0033332953612009,0.09928147,0.06083068,0.0018773065,0.0150757752051784
initial_test_model_train_1_mse_un_normed_819,0.0025511717948913,0.02019891,0.017245296,0.0022953874,0.0184069729191917
initial_test_model_train_1_mae_un_normed_819,0.0025511717948913,0.008949795,0.017924357,0.0015277565,0.011117033966503
initial_test_model_train_1_mse_auto_un_normed_819,0.0025511717948913,0.01661748,0.023324046,0.0022727137,0.018077825765121
initial_test_model_train_1_mae_auto_un_normed_819,0.0025511717948913,0.01921038,0.026371276,0.0015932783,0.0133012041965352
initial_test_model_train_1_mse_weighted_un_normed_819,0.0025511717948913,0.02396031,0.017433552,0.0014428322,0.0105735227800143
initial_test_model_train_1_mae_weighted_un_normed_819,0.0025511717948913,0.05570639,0.033623654,0.001536619,0.0111743186622935
initial_test_model_train_1_mse_all_normed_equal_26,0.0022785888717288,81.6289,106.62991,44.61915,648.4592096695557
initial_test_model_train_1_mae_all_normed_equal_26,0.0022785888717288,76.30707,93.465675,0.0026778064,0.0248079531963496
initial_test_model_train_1_mse_auto_all_normed_equal_26,0.0022785888717288,64.98445,126.38743,0.021507155,0.2139894162051867
initial_test_model_train_1_mae_auto_all_normed_equal_26,0.0022785888717288,79.09722,77.600685,1.4654297,14.109379338852689
initial_test_model_train_1_mse_weighted_all_normed_equal_26,0.0022785888717288,78.76948,111.990715,32.265385,379.1574546886495
initial_test_model_train_1_mae_weighted_all_normed_equal_26,0.0022785888717288,79.66537,82.44401,0.035006918,0.3501222361062018
initial_test_model_train_1_mse_all_normed_equal_292,0.0024398706087699,5.294276,51.672085,0.27403486,3.443380059552593
initial_test_model_train_1_mae_all_normed_equal_292,0.0024398706087699,2.2403786,45.37056,0.0031914944,0.0275604742482328
initial_test_model_train_1_mse_auto_all_normed_equal_292,0.0024398706087699,1.165443,18.272247,0.3987085,5.0344507139570585
initial_test_model_train_1_mae_auto_all_normed_equal_292,0.0024398706087699,4.5860133,38.80494,4.971159,164.46678339545505
initial_test_model_train_1_mse_weighted_all_normed_equal_292,0.0024398706087699,2.2612896,28.59477,0.006996757,0.0793268250635183
initial_test_model_train_1_mae_weighted_all_normed_equal_292,0.0024398706087699,5.1971273,29.700113,0.0015834557,0.0096737847628117
initial_test_model_train_1_mse_all_normed_equal_709,0.0033332953612009,11.032747,34.110615,0.09940721,1.2862855116607803
initial_test_model_train_1_mae_all_normed_equal_709,0.0033332953612009,9.838847,34.00307,0.0037384098,0.0344465638442955
initial_test_model_train_1_mse_auto_all_normed_equal_709,0.0033332953612009,17.928743,25.371922,2.2448635,31.298829385157205
initial_test_model_train_1_mae_auto_all_normed_equal_709,0.0033332953612009,3.1270287,20.396587,2.7208412,38.8029701708486
initial_test_model_train_1_mse_weighted_all_normed_equal_709,0.0033332953612009,15.926678,27.488775,4.0594916,58.428056091213726
initial_test_model_train_1_mae_weighted_all_normed_equal_709,0.0033332953612009,14.988754,20.620928,0.005313135,0.054652630256928
initial_test_model_train_1_mse_all_normed_equal_819,0.0025511717948913,14.399827,25.591928,16.135822,187.82429953987003
initial_test_model_train_1_mae_all_normed_equal_819,0.0025511717948913,2.3031292,26.200848,0.00136536,0.0101929785437489
initial_test_model_train_1_mse_auto_all_normed_equal_819,0.0025511717948913,18.85751,27.992,5.856058,83.55850162585655
initial_test_model_train_1_mae_auto_all_normed_equal_819,0.0025511717948913,19.211573,26.314507,3.3479586,41.85826638009394
initial_test_model_train_1_mse_weighted_all_normed_equal_819,0.0025511717948913,21.657652,24.26719,8.066945,101.35112437896258
initial_test_model_train_1_mae_weighted_all_normed_equal_819,0.0025511717948913,17.367176,22.938206,0.00393507,0.042265204975214
initial_test_model_train_1_mse_norm_independant_26,0.0011961019657907,0.0049580582,0.0034256496,0.0015505616,0.0152065389592393
initial_test_model_train_1_mae_norm_independant_26,0.0011961019657907,0.0012151877,0.0037270456,0.001145792,0.0025871085224912
initial_test_model_train_1_mse_auto_norm_independant_26,0.0011961019657907,0.0020398337,0.0054350593,0.0054442934,0.0526028498169093
initial_test_model_train_1_mae_auto_norm_independant_26,0.0011961019657907,0.0030801455,0.0030261038,0.0012662441,0.0043756491608514
initial_test_model_train_1_mse_weighted_norm_independant_26,0.0011961019657907,0.0019467479,0.00756809,0.0017673989,0.0116764936696796
initial_test_model_train_1_mae_weighted_norm_independant_26,0.0011961019657907,0.0025822502,0.002846243,0.0012290804,0.0039998214486942
initial_test_model_train_1_mse_norm_independant_292,0.0007871811770475,0.00091350556,0.0020260473,0.0022251918,0.028407169717215
initial_test_model_train_1_mae_norm_independant_292,0.0007871811770475,0.0016065714,0.0030563658,0.00089296047,0.0040703046087091
initial_test_model_train_1_mse_auto_norm_independant_292,0.0007871811770475,0.0011450619,0.019667218,0.0019831285,34.082030387692O9
initial_test_model_train_1_mae_auto_norm_independant_292,0.0007871811770475,0.0010934668,0.001836322,0.00086582475,0.0037245426584746
initial_test_model_train_1_mse_weighted_norm_independant_292,0.0007871811770475,0.0016333507,0.0013522677,0.00074015267,0.0023068853061613
initial_test_model_train_1_mae_weighted_norm_independant_292,0.0007871811770475,0.0009936265,0.008185517,0.00066462177,0.0014126245637193
initial_test_model_train_1_mse_norm_independant_709,0.0013523040625784,0.0019217136,0.0024671378,0.002207781,0.0156231856034059
initial_test_model_train_1_mae_norm_independant_709,0.0013523040625784,0.0032722908,0.0066146273,0.0013409376,0.0044673076445847
initial_test_model_train_1_mse_auto_norm_independant_709,0.0013523040625784,0.0022752637,0.0046433383,0.003192837,2.2783796073495
initial_test_model_train_1_mae_auto_norm_independant_709,0.0013523040625784,0.0037467072,0.004932948,0.0012102189,0.0030691102815455
initial_test_model_train_1_mse_weighted_norm_independant_709,0.0013523040625784,0.0024573018,0.0054264264,0.0023803653,0.4266954668752917
initial_test_model_train_1_mae_weighted_norm_independant_709,0.0013523040625784,0.0027572003,0.0030977894,0.0014283728,0.0066118515627934
initial_test_model_train_1_mse_norm_independant_819,0.0013459669702053,0.0018667441,0.0025834,0.0020893025,0.015444009751406
initial_test_model_train_1_mae_norm_independant_819,0.0013459669702053,0.0015246157,0.002860821,0.001246264,0.0029363863685595
initial_test_model_train_1_mse_auto_norm_independant_819,0.0013459669702053,0.0017528135,0.0024068444,0.0026540242,119.58206838752233
initial_test_model_train_1_mae_auto_norm_independant_819,0.0013459669702053,0.0031154016,0.004494887,0.0012447311,0.0026020427289336
initial_test_model_train_1_mse_weighted_norm_independant_819,0.0013459669702053,0.0015527194,0.0047899,0.0014635125,0.0059333323637953
initial_test_model_train_1_mae_weighted_norm_independant_819,0.0013459669702053,0.0024430535,0.0019534575,0.0012389397,0.0025586865973757
```

```
shallow_depthwise_diff_train_2_custom_mae_weighted_1e-3_norm_independant_292,0.0007871811770475,0.0012140871,0.0019004359,0.0007108505,0
medium_depthwise_diff_train_2_custom_mae_weighted_1e-3_norm_independant_292,0.0007871811770475,0.0026239287,0.0049702106,0.00080182496,0
complex_depthwise_diff_train_2_custom_mae_weighted_1e-3_norm_independant_292,0.0007871811770475,0.0012161265,0.001861764,0.000728415,0.0
shallow_depthwise_no_diff_train_2_custom_mae_weighted_1e-3_norm_independant_292,0.0007871811770475,0.0040164515,0.0060521206,0.001410398
medium_depthwise_no_diff_train_2_custom_mae_weighted_1e-3_norm_independant_292,0.0007871811770475,0.0010171345,0.0015427603,0.001338908,
complex_depthwise_no_diff_train_2_custom_mae_weighted_1e-3_norm_independant_292,0.0007871811770475,0.0020174128,0.0023822428,0.001293990
shallow_depthwise_diff_train_2_custom_mae_weighted_2e-3_norm_independant_292,0.0007871811770475,0.0013176344,0.0018620914,0.00071644306,
medium_depthwise_diff_train_2_custom_mae_weighted_2e-3_norm_independant_292,0.0007871811770475,0.0013114964,0.001970119,0.001040833,0.00
complex_depthwise_diff_train_2_custom_mae_weighted_2e-3_norm_independant_292,0.0007871811770475,0.0010078881,0.0019262928,0.00092487026,
shallow_depthwise_no_diff_train_2_custom_mae_weighted_2e-3_norm_independant_292,0.0007871811770475,0.0029459167,0.0034823976,0.00178692,
medium_depthwise_no_diff_train_2_custom_mae_weighted_2e-3_norm_independant_292,0.0007871811770475,0.0011773946,0.004069412,0.0013118087,
complex_depthwise_no_diff_train_2_custom_mae_weighted_2e-3_norm_independant_292,0.0007871811770475,0.002535391,0.0014929466,0.001265886,
shallow_depthwise_diff_train_2_custom_mae_weighted_1e-4_norm_independant_292,0.0007871811770475,0.0010463084,0.0012953219,0.00065196864,
medium_depthwise_diff_train_2_custom_mae_weighted_1e-4_norm_independant_292,0.0007871811770475,0.0012031557,0.008081451,0.0006574585,0.0
complex_depthwise_diff_train_2_custom_mae_weighted_1e-4_norm_independant_292,0.0007871811770475,0.0009946747,0.0039473544,0.0006596387,0
shallow_depthwise_no_diff_train_2_custom_mae_weighted_1e-4_norm_independant_292,0.0007871811770475,0.001432461,0.0058672223,0.0012556681
medium_depthwise_no_diff_train_2_custom_mae_weighted_1e-4_norm_independant_292,0.0007871811770475,0.0028155472,0.003378162,0.0012541681,
complex_depthwise_no_diff_train_2_custom_mae_weighted_1e-4_norm_independant_292,0.0007871811770475,0.0008352687,0.0019362184,0.001270282
shallow_depthwise_diff_train_2_custom_mae_weighted_1e-2_norm_independant_292,0.0007871811770475,0.0010828526,0.001186441,0.0026874994,0.
medium_depthwise_diff_train_2_custom_mae_weighted_1e-2_norm_independant_292,0.0007871811770475,0.0024146214,0.0013411109,0.0007857891,0.
complex_depthwise_diff_train_2_custom_mae_weighted_1e-2_norm_independant_292,0.0007871811770475,0.0022147049,0.0022183913,0.0016264247,0
shallow_depthwise_no_diff_train_2_custom_mae_weighted_1e-2_norm_independant_292,0.0007871811770475,0.001044421,0.002627327,0.001269793,0
medium_depthwise_no_diff_train_2_custom_mae_weighted_1e-2_norm_independant_292,0.0007871811770475,0.0019815825,0.0041559604,0.0018598847
complex_depthwise_no_diff_train_2_custom_mae_weighted_1e-2_norm_independant_292,0.0007871811770475,0.0034911972,0.0045222417,0.003248277
shallow_depthwise_diff_train_2_custom_mae_weighted_5e-4_norm_independant_292,0.0007871811770475,0.0017850442,0.0019192594,0.0006924935,0
medium_depthwise_diff_train_2_custom_mae_weighted_5e-4_norm_independant_292,0.0007871811770475,0.0011082024,0.0041309306,0.0006916095,0.
complex_depthwise_diff_train_2_custom_mae_weighted_5e-4_norm_independant_292,0.0007871811770475,0.0010818603,0.0027199981,0.00082565116,
shallow_depthwise_no_diff_train_2_custom_mae_weighted_5e-4_norm_independant_292,0.0007871811770475,0.0032600602,0.0041170977,0.001272883
medium_depthwise_no_diff_train_2_custom_mae_weighted_5e-4_norm_independant_292,0.0007871811770475,0.0015374693,0.0016186291,0.0012782164
complex_depthwise_no_diff_train_2_custom_mae_weighted_5e-4_norm_independant_292,0.0007871811770475,0.0049839746,0.004231788,0.0011886591
shallow_depthwise_diff_train_2_custom_mae_weighted_1e-3_more_epochs_norm_independant_292,0.0007871811770475,0.0012211158,0.0015724653,0.
medium_depthwise_diff_train_2_custom_mae_weighted_1e-3_more_epochs_norm_independant_292,0.0007871811770475,0.0011936337,0.0010920911,0.0
complex_depthwise_diff_train_2_custom_mae_weighted_1e-3_more_epochs_norm_independant_292,0.0007871811770475,0.0014842482,0.001256764,0.0
shallow_depthwise_no_diff_train_2_custom_mae_weighted_1e-3_more_epochs_norm_independant_292,0.0007871811770475,0.0010815947,0.0011247360
medium_depthwise_no_diff_train_2_custom_mae_weighted_1e-3_more_epochs_norm_independant_292,0.0007871811770475,0.0011103878,0.001212631,0
complex_depthwise_no_diff_train_2_custom_mae_weighted_1e-3_more_epochs_norm_independant_292,0.0007871811770475,0.0011131373,0.001815884
shallow_depthwise_diff_train_2_custom_mae_weighted_2e-3_more_epochs_norm_independant_292,0.0007871811770475,0.00095171493,0.0034905558,0
medium_depthwise_diff_train_2_custom_mae_weighted_2e-3_more_epochs_norm_independant_292,0.0007871811770475,0.0010097171,0.0013020947,0.0
complex_depthwise_diff_train_2_custom_mae_weighted_2e-3_more_epochs_norm_independant_292,0.0007871811770475,0.0011536635,0.0013057467,0.
shallow_depthwise_no_diff_train_2_custom_mae_weighted_2e-3_more_epochs_norm_independant_292,0.0007871811770475,0.0010489519,0.002893049,
medium_depthwise_no_diff_train_2_custom_mae_weighted_2e-3_more_epochs_norm_independant_292,0.0007871811770475,0.001108232,0.001099717,0.
complex_depthwise_no_diff_train_2_custom_mae_weighted_2e-3_more_epochs_norm_independant_292,0.0007871811770475,0.0010614918,0.0023558754
shallow_depthwise_diff_train_2_custom_mae_weighted_1e-4_more_epochs_norm_independant_292,0.0007871811770475,0.0009692639,0.0013932622,0.
medium_depthwise_diff_train_2_custom_mae_weighted_1e-4_more_epochs_norm_independant_292,0.0007871811770475,0.0017135999,0.002307287,0.00
complex_depthwise_diff_train_2_custom_mae_weighted_1e-4_more_epochs_norm_independant_292,0.0007871811770475,0.001362157,0.0019736108,0.0
shallow_depthwise_no_diff_train_2_custom_mae_weighted_1e-4_more_epochs_norm_independant_292,0.0007871811770475,0.0016818395,0.0028447155
medium_depthwise_no_diff_train_2_custom_mae_weighted_1e-4_more_epochs_norm_independant_292,0.0007871811770475,0.0015018563,0.0016320276,
complex_depthwise_no_diff_train_2_custom_mae_weighted_1e-4_more_epochs_norm_independant_292,0.0007871811770475,0.0012177814,0.0016827579
shallow_depthwise_diff_train_2_custom_mae_weighted_1e-2_more_epochs_norm_independant_292,0.0007871811770475,0.0011204095,0.0017276116,0.
medium_depthwise_diff_train_2_custom_mae_weighted_1e-2_more_epochs_norm_independant_292,0.0007871811770475,0.0012458056,0.0014732666,0.0
complex_depthwise_diff_train_2_custom_mae_weighted_1e-2_more_epochs_norm_independant_292,0.0007871811770475,0.0011783596,0.0021647033,0.
shallow_depthwise_no_diff_train_2_custom_mae_weighted_1e-2_more_epochs_norm_independant_292,0.0007871811770475,0.0011192788,0.0015508563
medium_depthwise_no_diff_train_2_custom_mae_weighted_1e-2_more_epochs_norm_independant_292,0.0007871811770475,0.00095041393,0.0010819425
complex_depthwise_no_diff_train_2_custom_mae_weighted_1e-2_more_epochs_norm_independant_292,0.0007871811770475,0.0010685748,0.0018726307
shallow_depthwise_diff_train_2_custom_mae_weighted_5e-4_more_epochs_norm_independant_292,0.0007871811770475,0.0007124605,0.002159733,0.0
medium_depthwise_diff_train_2_custom_mae_weighted_5e-4_more_epochs_norm_independant_292,0.0007871811770475,0.0009821771,0.0012895963,0.0
complex_depthwise_diff_train_2_custom_mae_weighted_5e-4_more_epochs_norm_independant_292,0.0007871811770475,0.0010757921,0.001089913,0.0
shallow_depthwise_no_diff_train_2_custom_mae_weighted_5e-4_more_epochs_norm_independant_292,0.0007871811770475,0.0009541946,0.0011291112
medium_depthwise_no_diff_train_2_custom_mae_weighted_5e-4_more_epochs_norm_independant_292,0.0007871811770475,0.0017969679,0.0011809501,
complex_depthwise_no_diff_train_2_custom_mae_weighted_5e-4_more_epochs_norm_independant_292,0.0007871811770475,0.001500612,0.0010938871,
medium_depthwise_diff_train_3_custom_mae_weighted_zw_09_nzw_01_norm_independant_292,0.0007871811770475,0.0012042733,0.0011374223,0.0006
medium_depthwise_diff_train_3_custom_mae_weighted_zw_08_nzw_02_norm_independant_292,0.0007871811770475,0.0010420616,0.0018296761,0.0006
medium_depthwise_diff_train_3_custom_mae_weighted_zw_07_nzw_03_norm_independant_292,0.0007871811770475,0.001275608,0.0018446102,0.000649
medium_depthwise_diff_train_3_custom_mae_weighted_zw_06_nzw_04_norm_independant_292,0.0007871811770475,0.0012484898,0.0010507375,0.0006
medium_depthwise_diff_train_3_custom_mae_weighted_zw_05_nzw_05_norm_independant_292,0.0007871811770475,0.00089357485,0.0015171864,0.0006
medium_depthwise_diff_train_3_custom_mae_weighted_zw_04_nzw_06_norm_independant_292,0.0007871811770475,0.001126965,0.0020285249,0.000649
medium_depthwise_diff_train_3_custom_mae_weighted_shuffle_on_norm_independant_292,0.0007871811770475,0.0021654107,0.0012465274,0.0006476
medium_depthwise_diff_train_3_custom_mae_weighted_shuffle_off_norm_independant_292,0.0007871811770475,0.0015784751,0.0013071987,0.000649
medium_depthwise_diff_train_3_custom_mae_weighted_shuffle_on_norm_independant_normal_292,0.0007871811770475,0.0010765365,0.0014090019,0.
medium_depthwise_diff_train_3_custom_mae_weighted_shuffle_on_norm_independant_masked_292,0.0007871811770475,0.0013812446,0.0037139733,0.
medium_depthwise_diff_train_3_custom_mae_weighted_shuffle_on_norm_independant_rf_added_292,0.0056286526851929,0.0059614927,0.005841841,0
medium_depthwise_diff_train_4_custom_mae_weighted_1_norm_independant_normal_292,0.0007871811770475,0.0011634432,0.0013362505,0.000668780
medium_depthwise_diff_train_4_custom_mae_weighted_2_norm_independant_normal_292,0.0007871811770475,0.00095762586,0.00163615,0.0007257050
medium_depthwise_diff_train_4_custom_mae_weighted_3_norm_independant_normal_292,0.0007871811770475,0.0012843743,0.0014467528,0.000658170
medium_depthwise_diff_train_4_custom_mae_weighted_4_norm_independant_normal_292,0.0007871811770475,0.0010665995,0.002771737,0.0016418807
medium_depthwise_diff_train_4_custom_mae_weighted_5_norm_independant_normal_292,0.0007871811770475,0.00094230904,0.0023979368,0.0006572
medium_depthwise_diff_train_4_custom_mae_weighted_6_norm_independant_normal_292,0.0007871811770475,0.0009898654,0.0020259449,0.000648414
medium_depthwise_diff_train_4_custom_mae_weighted_1_norm_independant_masked_292,0.0007871811770475,0.0021155362,0.0013054012,0.001245632
medium_depthwise_diff_train_4_custom_mae_weighted_2_norm_independant_masked_292,0.0007871811770475,0.0016933961,0.001607783,0.0006481260
medium_depthwise_diff_train_4_custom_mae_weighted_3_norm_independant_masked_292,0.0007871811770475,0.0015447089,0.0022196025,0.0006732610
medium_depthwise_diff_train_4_custom_mae_weighted_4_norm_independant_masked_292,0.0007871811770475,0.0013100905,0.0018047951,0.002807074
medium_depthwise_diff_train_4_custom_mae_weighted_5_norm_independant_masked_292,0.0007871811770475,0.00138857,0.002935844,0.00072624057,
medium_depthwise_diff_train_4_custom_mae_weighted_6_norm_independant_masked_292,0.0007871811770475,0.0014165496,0.0017291179,0.0006478107
medium_depthwise_diff_train_4_custom_mae_weighted_1_norm_independant_rf_added_292,0.0056286526851929,0.0055744513,0.0060025216,0.005569
medium_depthwise_diff_train_4_custom_mae_weighted_2_norm_independant_rf_added_292,0.0056286526851929,0.006019837,0.0051540625,0.0056694
medium_depthwise_diff_train_4_custom_mae_weighted_3_norm_independant_rf_added_292,0.0056286526851929,0.005613463,0.0059447484,0.0055167
medium_depthwise_diff_train_4_custom_mae_weighted_4_norm_independant_rf_added_292,0.0056286526851929,0.005625528,0.006826772,0.0055186
medium_depthwise_diff_train_4_custom_mae_weighted_5_norm_independant_rf_added_292,0.0056286526851929,0.0062357564,0.005204452,0.0055303
medium_depthwise_diff_train_4_custom_mae_weighted_6_norm_independant_rf_added_292,0.0056286526851929,0.006033032,0.005555527,0.0055261
medium_depthwise_diff_train_4_custom_mae_weighted_7_20 epochs_norm_independant_normal_292,0.0007871811770475,0.0010415805,0.0010247573,0
medium_depthwise_diff_train_4_custom_mae_weighted_7_20 epochs_norm_independant_masked_292,0.0007871811770475,0.0010366783,0.0028363706,0
medium_depthwise_diff_train_4_custom_mae_weighted_7_20 epochs_norm_independant_rf_added_292,0.0056286526851929,0.005598067,0.0051983437,
simple_incep_train_5_custom_mae_weighted_8_norm_independant_normal_292,0.0007871811770475,0.0009662657,0.001178633,0.00066078466,0.0014
medium_incep_train_5_custom_mae_weighted_8_norm_independant_normal_292,0.0007871811770475,0.0010319747,0.0010540975,0.0006755931,0.0016
simple_incep_train_5_custom_mae_weighted_9_norm_independant_normal_292,0.0007871811770475,0.0010930636,0.0012467903,0.00064848975,0.0012
medium_incep_train_5_custom_mae_weighted_9_norm_independant_normal_292,0.0007871811770475,0.0011658908,0.0010638067,0.0006324215,0.0014
simple_incep_train_5_custom_mae_weighted_10_norm_independant_normal_292,0.0007871811770475,0.0013660941,0.0011458627,0.00063410465,0.001
medium_incep_train_5_custom_mae_weighted_10_norm_independant_normal_292,0.0007871811770475,0.0010114962,0.0013619313,0.0006476477,0.0012
```

simple_incep_train_5_custom_mae_weighted_8_norm_independant_masked_292,0.0007871811770475,0.0009861486,0.0013325573,0.00081108493,0.0030515878
medium_incep_train_5_custom_mae_weighted_8_norm_independant_masked_292,0.0007871811770475,0.0014270117,0.0038307488,0.00065428525,0.0013415804
simple_incep_train_5_custom_mae_weighted_9_norm_independant_masked_292,0.0007871811770475,0.0014686966,0.0013899022,0.0006478273,0.00126191196
medium_incep_train_5_custom_mae_weighted_9_norm_independant_masked_292,0.0007871811770475,0.001183265,0.0017192628,0.0006472916,0.001253807562
simple_incep_train_5_custom_mae_weighted_10_norm_independant_masked_292,0.0007871811770475,0.00089728617,0.0018781321,0.00064831576,0.00125346
medium_incep_train_5_custom_mae_weighted_10_norm_independant_masked_292,0.0007871811770475,0.001222065,0.0011971091,0.00064968894,0.0012668934
simple_incep_train_5_custom_mae_weighted_8_norm_independant_rf_added_292,0.0056286526851929,0.005676362,0.0053045363,0.005516845,0.00615931651
medium_incep_train_5_custom_mae_weighted_8_norm_independant_rf_added_292,0.0056286526851929,0.00565464,0.005513255,0.0055341884,0.006158715813
simple_incep_train_5_custom_mae_weighted_9_norm_independant_rf_added_292,0.0056286526851929,0.0062925927,0.0052308906,0.0055141505,0.006159418
medium_incep_train_5_custom_mae_weighted_9_norm_independant_rf_added_292,0.0056286526851929,0.006006147,0.005281879,0.005513752,0.006159585619
simple_incep_train_5_custom_mae_weighted_10_norm_independant_rf_added_292,0.0056286526851929,0.005750453,0.0051826066,0.0055185943,0.006158993
medium_incep_train_5_custom_mae_weighted_10_norm_independant_rf_added_292,0.0056286526851929,0.0058884546,0.005650408,0.0055143177,0.006159585

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] Ioannis Karafyllidis and Adonios Thanailakis. "A model for predicting forest fire spreading using cellular automata". In: *Ecological Modelling* 99.1 (1997), pp. 87–97.

[12] Lemont B Kier, Paul G Seybold, and Chao-Kun Cheng. *Modeling chemical systems using cellular automata*. Springer Science & Business Media, 2005.

[13] Dieter A Wolf-Gladrow. *Lattice-gas cellular automata and lattice Boltzmann models: an introduction*. Springer, 2004.

[14] Rahim Alizadeh. "A dynamic cellular automaton model for evacuation process with obstacles". In: *Safety Science* 49.2 (2011), pp. 315–323.

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

[16] Pravallika Etoori, Manoj Chinnakotla, and Radhika Mamidi. "Automatic spelling correction for resource-scarce languages using deep learning". In: *Proceedings of ACL 2018, Student Research Workshop*. 2018, pp. 146–152.

[17] Malay Haldar et al. "Applying deep learning to airbnb search". In: *proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & Data Mining*. 2019, pp. 1927–1935.

[18] Abhishek Gupta et al. "Deep learning for object detection and scene perception in self-driving cars: Survey, challenges, and open issues". In: *Array* 10 (2021), p. 100057.

[19] Shaveta Dargan et al. "A survey of deep learning and its applications: a new paradigm to machine learning". In: *Archives of Computational Methods in Engineering* 27 (2020), pp. 1071–1092.

[20] 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.

[21] 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.

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

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

[24] 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.

[25] 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.

[26] 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.

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

[28] 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.

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

[30] 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.

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

[32] Xiaohan Ding et al. "Diverse branch block: Building a convolution as an inception-like unit". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 10886–10895.

[33] Bikesh Kumar Singh, Kesari Verma, and AS Thoke. "Investigations on impact of feature normalization techniques on classifier's performance in breast tumor classification". In: *International Journal of Computer Applications* 116.19 (2015).

[34] Aryan Jadon, Avinash Patil, and Shruti Jadon. "A Comprehensive Survey of Regression Based Loss Functions for Time Series Forecasting". In: *arXiv preprint arXiv:2211.02989* (2022).