SHALLOW WATER EMULATION WITH NEURAL NETWORKS FOR DISASTER PREVENTION

Diego Calanzone
Università degli Studi di Parma
diego.calanzone@studenti.unipr.it

ABSTRACT

Shallow water emulations allow to study fluid dynamics related to natural disasters. Currently, Physics Simulators based on Numeric Calculus (ie: LISFLOOD-FP) are implemented in research. However, for real-time prediction the computational cost becomes unsustainable with the increasing size of the environment to simulate and an increasing number of factors involved. As Deep Learning algorithms have become popular for a variety of prediction tasks, we study various applications of statistical and machine learning models to tackle this challenge. In physics, we consider two classes of simulations for fluids:

- **Smoothed Particle Hydrodynamics (SPH)**: in a three-dimensional space, the interaction between fluids and objects is modeled as reactions between particles of different nature.
- **Shallow Water Emulations (SWE)**: in a two-dimensional space, water depths are analyzed in comparison to floor elevation and the intensity of the stream.

This study focuses on *Shallow Water Emulations*. The aim is to provide a framework for simulating flash floods with three different settings:

1. Predicting a sequence of **N** future states of water depths
2. Classifying patches of images as "at risk" in the future
3. Predicting one single future state of water depths

1. STATE OF THE ART

For this class of tasks, several techniques have been used in Fluid dynamics, ranging from parallel computing to machine learning.

- **Data Model:** we based our data representation on the work of *Dr. Alessandro Dal Palú (2014) [1]*: one single state is represented by a two-dimensional grid, where a water depth is given at an *X, Y* location. This grid includes multiple levels of resolution and supports domain decomposition for parallel computing on GPU.

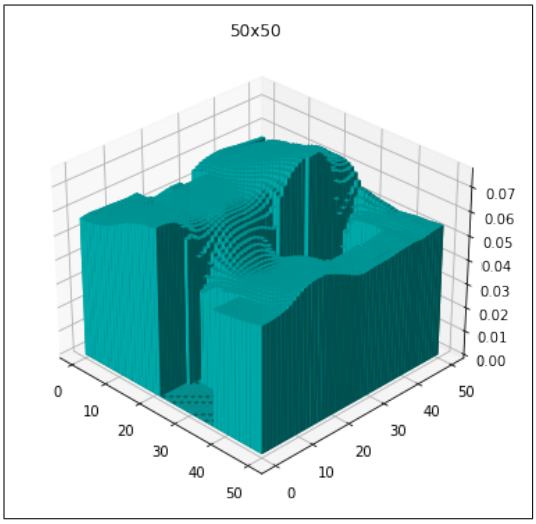

Figure 1. 3D Visualization of a DEP map in a certain region (water depths)

- **Introduction of Neural Networks:** machine learning techniques have been applied for shallow water emulations. *M. Varesi (2018) [2]* implemented a feed-forward network to perform multi-variate regression on time-series sampled from the Secchia River dataset (Emilia Romagna, Italy). Despite the discrete results, this setting was not suited for bidimensional and multi-modal data.
- **Convolutional Neural Networks:** basing on the simulator *LISFLOOD-FP*, *S. Kabir, S. Patidar, X. Xia e Q. Liang (2020) [3]* designed an architecture based on Convolutional Blocks to process images. In their paper, CNNs are compared to Support Vector Machines on this task, with interesting results in predicting a short sequence of future states in real scenarios.
- **The PARFLOOD Simulator:** *F. Vignali [4]* introduces the *PARFLOOD* simulator, developed at the University of Parma and designed to compute Navier-Stokes equations in parallel.
- **Next frame prediction:** *C. Vondrick e A.Torralba (2016) [5]* propose a novel architecture trained to predict transformations in the future frames using skip connections.

- **Vision Transformers:** Google Brain Research fully replaces convolutions with *Attention [7]* Blocks encoding patches of images as tokens for classification tasks *A. Dosovitskiy, N. Houlsby (2020) [6].* This novel architecture is emerging in current research, underlining the potential of models free of inductive-bias, with much higher receptivity.

2. EXPERIMENTAL ENVIRONMENT

Type	Hardware	Amount
Personal	NVIDIA GTX 1070	1
Cloud	NVIDIA Tesla K80	1
Academic	NVIDIA Tesla P100	7
Cloud	Google TPU TF 2.1	1

Frameworks: Tensorflow 2, PyTorch 1.71, Statsmodels 0.12.1, Numpy 1.19.5, Matplotlib 3.3.3

3. DATA GENERATION

We trained our models to learn from the simulations computed by PARFLOOD [14].

We provide as input to the simulator **S** an initial enviroment described by different maps: floor elevation, water depths, stream intensity over time, position of physical boundaries. In this study, we will focus on the Toce River Dataset generated by the HyLab team from samples registered in Piemonte, Italy.

- BTM Map: bathymetry, river floor elevation
- MAN File: roughness/diameter rate for the water conduit, considering the Manning coefficient
- INH Map: map of starting conditions (surface, water depths, stream intensity)
- BLN Map: map of physical boundaries (borders, solids, surfaces)
- BCC File: time series of the stream intensity
- Configuration File: duration of the simulation (hours), sampling rate (hours), additional parameters for approximation (default values will be considered)

Given the *X, Y* location for the stream **b** (with a certain intensity at source), the PARFLOOD model computes a simulation of $\Delta\mathbf{t}$ hours with sampling rate **g**, generating a sequence of **N** states:

$$N = \frac{\Delta t}{g} \tag{1}$$

In our experiments: $\Delta\mathbf{t} = 1$, $\mathbf{g} = 0{,}005$.

Each state comprises 3 maps: *DEP* for water depths, *VVX* for stream velocities in the x-axis, *VVY* for stream velocities in the y-axis. Eventually the simulator generates the *MAXWSE* map, containing the maximum water depths reached during the simulation for each cell, this will become relevant for the risk classification task.

4. PARALLEL COMPUTING

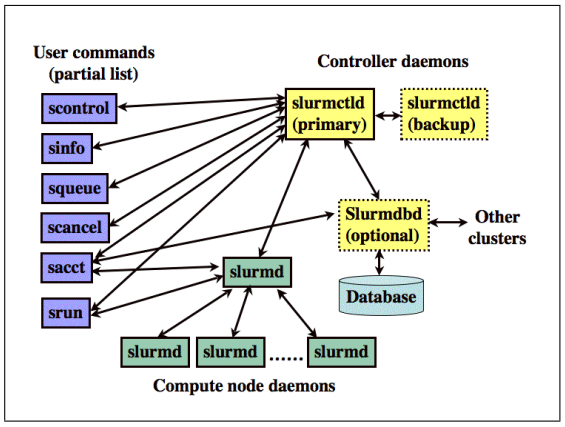

Figure 2. Slurm Architecture

To efficiently generate a multitude of scenarios for simulation, we relied on Slurm, a known workload manager for High Performance Computing.
By modulating the stream intensity variations over time, we have generated **100** different conditions of flooding to achieve a sufficiently vary dataset for our model.

The data generation process can be summarized in the following steps

1. Generating 100 sequences of stream intensities with normal distribution and random maxima
2. Submission for 100 PARFLOOD simulations as Slurm Jobs
3. Decoding of the generated states from the model from binary to float32
4. Data preprocessing: data cleansing, normalization, outliers analysis

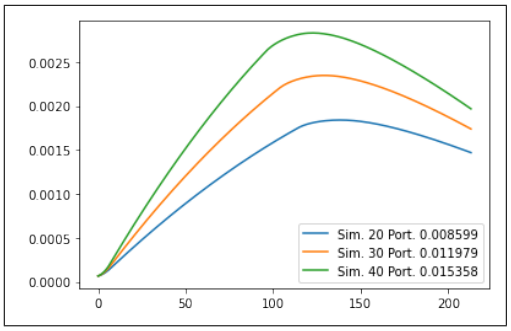

Figure 3. Average water depths over time with varying stream intensities (port.)

5. DATA UNDERSTANDING

From domain knowledge we have identified the nature of the considered variables:

- Stream Intensity **B** [m^3/s]: *feature*, continuous
- Bathymetry **BTM** [m]: *feature*, continuous
- Depth **DEP** [m]: *feature, target*,continuous
- Risk Matrix **BIN**: *target*, dichotomous

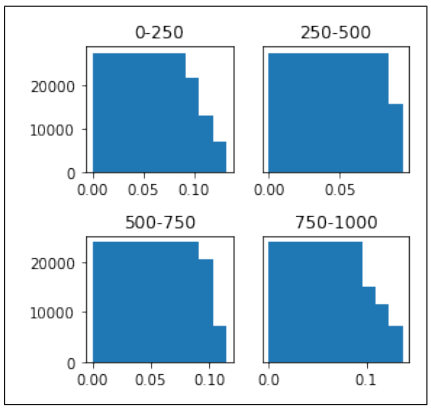

Figure 4. DEP distribution over 4 ranges of frames.

Univariate Analysis: firstly, *BTM* values are constant over time, since we don't consider changes in the floor surface.

- Water depths (meters) range from **0.00** m e **0.14** m. The simulation can be split in various phases represented by the variations of water depths: inflow, stabilization, outflow. (**Figure 4**).
- The maximum stream intensity for each simulation varies from **0.018** m^3/s to **0.18** m^3/s. This allowed us to consider floodings of various intensities.

This early analysis brought various questions regarding the information to feed in the models to correctly understand the progression of states (**Figure 6**):

- Inflow phase: the stream of water hits the surfaces and starts flooding the river basin. The liquid accumulates close to the obstacles (houses, pillars, rocks), hence the water levels temporarily rise.
- Stabilization phase: the stream is completely distributed on the surface, the average height of water columns decreases and no major impacts with obstacles are registered.
- Outflow phase: the stream progressively decreases, thus the water starts draining and basing on the bathymetry, some water depths will become stable due to the presence of pits.

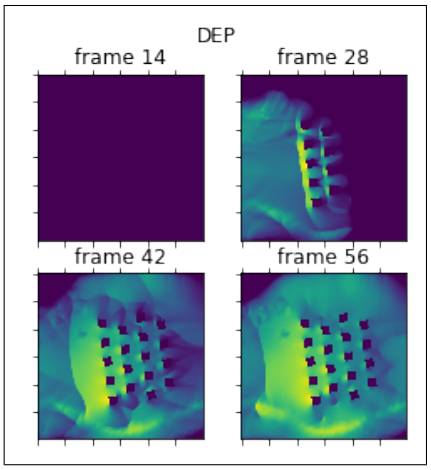

Figure 5. Visualization of water depths over four ranges of time.

6. DATA PREPROCESSING

Basing on the three tasks previously introduced (1, 2, 3), we have processed the dataset with different techniques for each experiment:

- **Task (1, 2, 3)**: data standardization with μ the mean value and σ the standard deviation:

$$x = \frac{(x - \mu)}{\sigma} \tag{2}$$

- **Task (2)**: discretization of the *MAXWSE* map to generate the target binary matrix *BIN*. We have set a threshold of **0.001 m** and **0.005 m**, any cell with a higher level of water depth is considered "flooded" or "at risk".
- **Task (1)**: data augmentation. For higher sampling rates, the simulator generates frames temporally distant from each other. To smoother the variation between these states, we introduce a simple interpolation formula. Let $\alpha = 0.5$ be the learning rate, x_i the frame at time *i* and x_{i+1} its successor, the intermediate frame is defined as following:

$$x_{\frac{i+1}{2}} = (1 - \alpha) * x_i + \alpha * x_{i+1} \tag{3}$$

  However, in our experiments we identified an upper bound of **3** iterations, beyond which the difference between consecutive frames becomes too small for the model to distinguish, resulting in poor predictive power.
- **Task (1)**: frame cropping, we have limited the scope of prediction to a central area of 300x300 pixels, where major collisions occur between the stream and various buildings.
- **Task (1, 3)**: data cleansing, the simulator uses "flag values" to indicate the presence of solid objects or surfaces out of the river basins. With domain knowledge, we have replaced any value above a set threshold of x_{DEP} >= **1.E+5** with zeros.

- **Task (1)**: application of the Dicky-Fuller test to achieve stationarity in time series through differentiation. This allowed us to perform univariate regression with ARIMA.

We have adopted the **Holdout Strategy** to perform effective training of our model, hyperparameter tuning has been carried out through automated **Grid Search** with the Weights Biases framework.

7. MODELING

- **Task 1 - ARIMA** (Auto Regressive Integrated Moving Average): to predict a sequence of water depths, we have firstly implemented an autoregressive statistical model, where the input time series is by definition the target for prediction. In its simplest version, ARIMA handles only one variable (first limitation), thus we have reduced the task to predicting the average water depth in the future. Let $N_0 \rightarrow$i be the number of past frames, the target is the sequence of $N_{i+1} \rightarrow$i+w, where **w** is the prediction window size.

- **Task 1 - Convolutional Auto Encoder**: moving on machine learning models, to predict a sequence of images (maps) we have designed a *2D Convolutional Autoencoder* with LSTM layers for the latent space. The bottleneck network is a simple CNN encoder, even though multiple tests have been carried out with residual blocks as well (slight improvements in performance). Each convolutional block comprises an *Average Pooling* and a *Batch Normalization* layer.

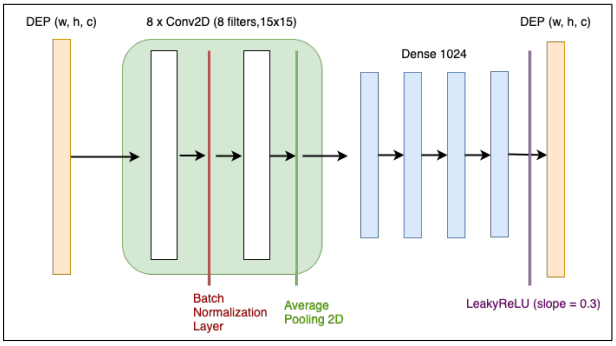

Figure 6. Convolutional Autoencoder (CNN + Dense)

We have tested out a decoder built with Convolutional Transpose Layers, however increasingly high computatonal costs leaves out this digression for future experiments.

- **Task 2 - Risk Classification**: as pictured in **Figure 7**, we have build a feed forward neural network to predict the areas at risk of flooding. The *BTM* map is concatenated to the stream intensity (fluid conduit) B, batch normalization is applied after each dense layer. Eventually, the **sigmoid** activation is applied to predict probabilities:

$$S(x) = \frac{1}{1 + e^{-x}} \tag{4}$$

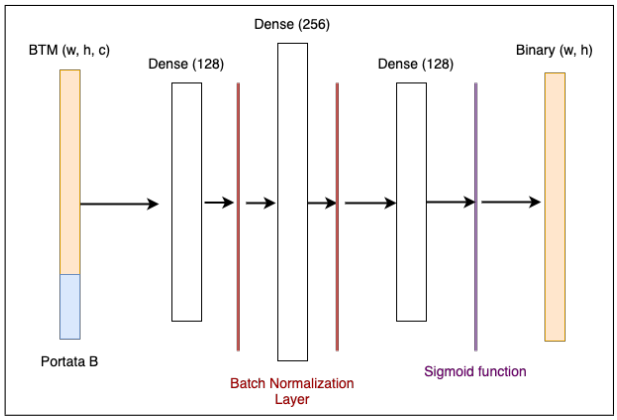

Figure 7. Feed Forward Classification Network

- **Task 3 - Single state prediction**: for this task, a more complex architecture has been proposed. Different inputs are processed with different branches: the bathymetry *BTM* propagates through a convolutional network with the simple CNN blocks defined previously; the stream intensity time series **B** is fed into three LSTM layers, a hidden representation is then concatenated with the encoded BTM map and fed into a stack of Dense layers.

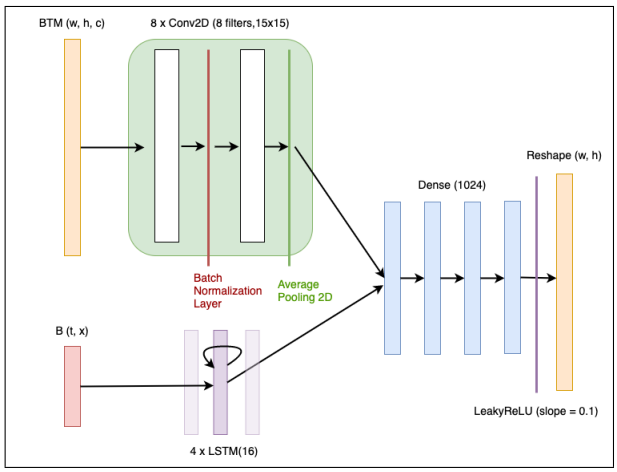

Figure 8. CNN + LSTM Autoencoder

8. EVALUATION

Several experiments have been conducted to identify a reliable loss function. From domain knowledge, we chose *Relative Error (RE)* and *Cosine Similarity* for training and validation.

- Relative Error (RE):

$$\delta = \frac{(Y - \widehat{Y})}{Y} * 100 \tag{5}$$

Task 1 - Sequence prediction

Model	Metric	Val. Score
Univariate ARIMA	Relative Error (RE)	72,5%
CNN Autoencoder	Relative Error (RE)	18,2%

Task 2 - Risk classification

Model	Metric	Val. Score
Feed Forward$_1$	Cosine Similarity	70,9%
Feed Forward$_2$	Cosine Similarity	62,8%

[1] Image resolution: 33%, 3 layers 128x256x128
[2] Image resolution: 100%, 3 layers 512x1024x512

Task 3 - Future state prediction

Model	Metric	Val. Score
CNN + LSTM	Relative Error (RE)	71,9%

For Hyperparameter Tuning, we perfomed an *Automated Grid Search* on Task **2** and **3**.

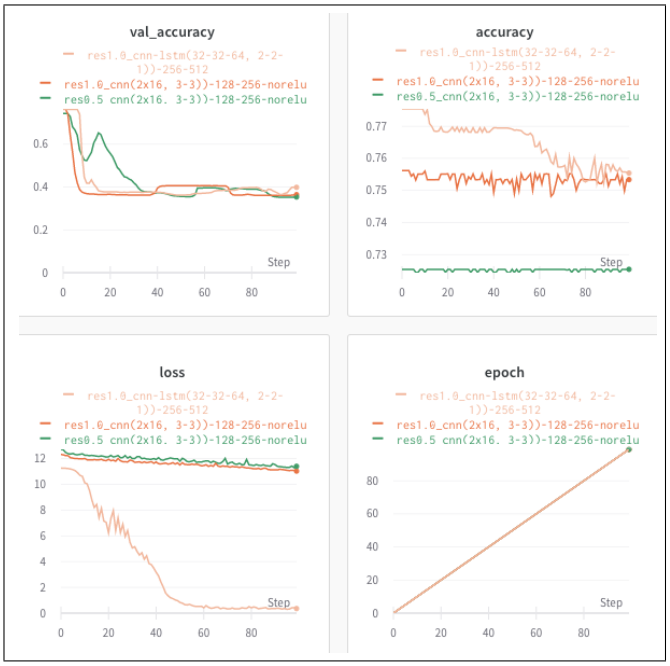

Figure 9. A comparison between variants of architectures for Task 3 (source: Weights Biases).

9. CONCLUSIONS

- **Next Frame Prediction Task**, the models considered in this study have relevant limitations: ARIMA is a univariate autoregressive model which cannot process data on two dimensions (images, matrices); the CNN autoencoder poorly performs in predicting the following frames, predictions are often blurry, suggesting the model may "collapse" into an approximate solution averaging all the possible outcomes. This issue is very common with deterministic models, suggesting further research for non-deterministic solutions.
- **Inductive Biases**: Convolutional Neural Networks are designed to process images. With shared weights, a sliding kernels extract features such as edges and visual patterns. However, it is questioned whether these receptive fields are excessively small to detect relationships between pixels from opposite sides of an image. Recent research in Natural Language Processing revealed the potential of a novel computational block: the Transformer [7] . By removing any inductive bias, Transformers blocks compute attention matrices. Vision Transformers [6], from Google Research, propose a transformer-based encoder to convert patches of images into tokens. This novel technique leaves space for further research as an alternative to Convolution-Based methods.
- **Single Future State Prediction**: considering the difficulties encountered in Task 1, we have focused part of this study on predicting one single future state given the past sequence. The CNN + LSTM autoencoder performs discretely well at native image resolution. The LeakyReLU activation with a negative slope of **0.5** best suited to output water depths without clipping values close to zero. With further hyperparameter tuning, we concluded that dense layers with over 1024 or 2048 neurons allowed to predict neat images without any "glitch" or distorsion.

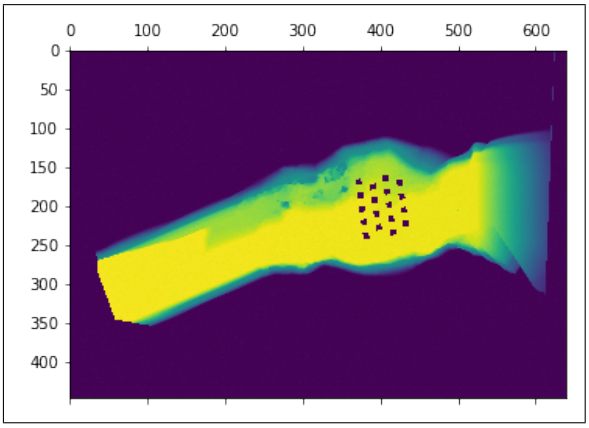

Figure 10. Generated DEP map in Task 3 (outflow phase).

In conclusion, we can conclude that Neural Networks deserve further studies with aim of approximating burdensome Shallow Water Emulations. Uncertainty is an occurring issue in video prediction tasks, since multiple outcomes are possible and deterministic models are likely to fail in taking into account this factor. With increasingly large scenarios, local receptivity in CNNs results extremely limiting in determining the correlation in larger sequences of frames, representing an extended geographical area.

10. REFERENCES

[1] *GPU-enhanced Finite Volume Shallow Water solver for fast flood simulations*.
R. Vacondio, A. Dal Palù, P. Mignosa (2014)
http://bit.ly/3q4T1H1

[2] *Apprendimento e predizione di dati fluviali tramite reti neurali*.
M. Varesi (2018)
http://bit.ly/3rNGtEv

[3] *A deep convolutional neural network model for rapid prediction of fluvial flood inundation*.
Syed Rezwan Kabir, S. Patidar, Xilin Xia, Qiuhua Liang (2020)
https://bit.ly/3cROWlG

[4] *Simulazioni ad alta risoluzione degli allagamenti-generati da brecce arginali in zone urban*.
F. Vignali (2020)
http://bit.ly/3rNGtEv

[5] *Generating the Future with Adversarial Transformers*.
C. Vondrick, A. Torralba (2020)
https://bit.ly/3p89GIr

[6] *An image is worth 16x16 words: Transformers for Image Recognition at scale*.
A. Dosovitskiy, N. Houlsby (2020)
https://bit.ly/3rJpIu1

[7] *Attention is All You Need*.
A. Vaswani (2017)
https://bit.ly/3q7MueC

[8] *Tensorflow.org Documentation*.
https://www.tensorflow.org/api_docs

[9] *Keras.org Documentation*.
https://keras.io/api/

[10] *PyTorch.org Documentation*.
https://pytorch.org/docs/stable/

[11] *Pierian Data - Education*.
Data Science Bootcamp
Neural Networks with Tensorflow Bootcamp
https://www.pieriandata.com/

[12] *Weights & Biases Documentation*.
https://github.com/wandb

[13] *Equazioni di Navier Stokes - Wikipedia*.
http://bit.ly/36ZZZFF

[14] *Laboratorio HyLab - Modello PARFLOOD*.
http://bit.ly/3tH7VoZ

[15] *ARIMA Model – Complete Guide to Time Series Forecasting in Python*
Marchinelearningplus.org
http://bit.ly/3tJB3vS

[16] *ARIMA Model Python Example — Time Series Forecasting*.
Towardsdatascience.com
http://bit.ly/3q8nK62