Physics-Aware Downsampling with Deep Learning for Scalable Flood Modeling

Niv Giladi[1,2] **Zvika Ben-Haim**[1] **Sella Nevo**[1] **Yossi Matias**[1] **Daniel Soudry**[2]

[1]Google Research
[2]Technion - Israel Institute of Technology

{giladiniv, daniel.soudry}@gmail.com
{zvika, sellanevo, yossi}@google.com

Abstract

Background. Floods are the most common natural disaster in the world, affecting the lives of hundreds of millions. Flood forecasting is therefore a vitally important endeavor, typically achieved using physical water flow simulations, which rely on accurate terrain elevation maps. However, such simulations, based on solving partial differential equations, are computationally prohibitive on a large scale. This scalability issue is commonly alleviated using a coarse grid representation of the elevation map, though this representation may distort crucial terrain details, leading to significant inaccuracies in the simulation.
Contributions. We train a deep neural network to perform physics-informed downsampling of the terrain map: we optimize the coarse grid representation of the terrain maps, so that the flood prediction will match the fine grid solution. For the learning process to succeed, we configure a dataset specifically for this task. We demonstrate that with this method, it is possible to achieve a significant reduction in computational cost, while maintaining an accurate solution. A reference implementation accompanies the paper as well as documentation and code for dataset reproduction.

1 Introduction

A physical process is a time-evolving phenomenon marked by gradual transitions through a series of states, following physical rules. Scientists attempt to model these physical processes by employing analytical descriptions that correspond to the underlying physics of these processes. Commonly, the analytical descriptions, or physical models, such as conservation laws, are in the form of partial differential equations (PDEs). Physical models are the backbone behind modeling phenomena such as weather forecasts, flood forecasting, aerodynamics, chemical processes, and more. Traditionally, these physical models are solved numerically, as no closed form solution is available for most problems. One typical numerical method is to discretize the PDEs using a Taylor expansion and repeatedly integrate in time over short intervals. However, solving PDEs numerically on a large scale is challenging because of the need to resolve spatiotemporal features over many scales.

In this paper, we focus on hydraulic models, which are a widely used physical models. The goal of these models is to simulate the flow of water in a given environment. We are specifically interested in simulating water flow through a floodplain in order to predict inundation, i.e., where and to what extent will flooding occur. Specifically, in inundation modeling we are given an elevation map of the terrain, boundary conditions, initial conditions — and we wish to predict, the water height in each pixel at some future time t, with maximum accuracy and minimum computational cost. Such prediction is an important task in operational flood alert systems [5, 43, 10].

35th Conference on Neural Information Processing Systems (NeurIPS 2021), Sydney, Australia.

Unfortunately, inundation modeling is often computationally intractable when modeling flow on a large scale with fine grid resolution. A coarse grid representation of the elevation map can significantly improve computation time, for two reasons. First, less computation is done at each step of the PDE solver. Specifically, in a discretized PDE the per-step spatial computation is done between neighboring pixels. Therefore, if we downsample a d-dimensional map by a factor of x, we decrease this per-step computation by a factor of x^d. Second, the maximum stable time step in the numerical solution can be increased by a factor of x, as entailed by the Courant–Friedrichs–Lewy (CFL) condition [23, 7]. Taking both factors into account, we gain a total theoretical speedup of x^{d+1}. For example, in this paper we will show that for the two-dimensional problem ($d = 2$) of inundation modeling, we retain fairly good accuracy when downsampling by a factor of 16, leading to a significant $4096\times$ speedup.

So far it was considered challenging to coarse-grain a general elevation map without losing accuracy in the inundation prediction. In particular, small details in the elevation map can significantly impact the water flow evolution. Narrow canals, dams, and embankments (barriers preventing a river from flooding an area) are examples of such details that require special care. There are several typical methods to represent an elevation map on a coarse grid, all of which can cause a sub-optimal representation, as small, important details might get lost [11, 44, 43]. For example, when applying standard downsampling methods, an 8 meter wide embankment might be reduced to half its original height above its neighboring surroundings in a 16 meters resolution coarse map. More complex downsampling techniques can be employed, but in practice, to achieve high-quality simulations at a coarse resolution, domain expertise is required, along with a deep understating of the underlying physics of the specific problem.

To address these limitations, we optimize the coarse grid representation of the elevation map so that its water height solution will match the accurate fine grid solution. This is accomplished by combining a deep neural network (DNN) with the PDEs describing the flow of water. The DNN downsamples the fine grid elevation map, and the downsampled elevation map is fed into the PDEs, where each time step is a recurrent unit, to calculate the water height solution. Thus, the DNN is optimized directly on the water height solution, through the PDEs. This also requires back-propagating through the PDEs, whose time-steps need to be unrolled. We demonstrate that this approach can yield a better coarse grid representation than conventional methods, in terms of water flow estimation. To this end, we configured a dataset designed specifically for inundation modeling. Each sample consists of a fine grid elevation map, boundary conditions, and a simulation time. The ground truth of each sample is the corresponding fluid state, calculated on a fine grid. This dataset is available along with code for further use.

The contributions of our work include:

- A new framework for learning how to coarse-grain a PDE together with its external environment — by minimizing the distance between the solutions on the fine and coarse grids (Sec. 4.1).
- A novel configured dataset for 2D water flow estimation, with accessible code for reproduction[1]. (Sec. 4.2)
- A demonstration that gradients backpropgated for *tens of thousands* of steps through the PDE remain informative (Sec. 5.1) and can pinpoint key features (Fig. 2)
- A demonstration of the accuracy of our approach in inundation modeling, for two different application settings (Sec. 5.2 and 5.3). Since we decrease the map resolution by a factor 16, this entails a $4096\times$ speedup. Such a speedup has a significant impact on the feasibility of real-time flood warning for large populations (Sec. 8).

2 Related Work

The integration of machine learning with physical models, described by PDEs, has emerged in recent years as a useful tool for efficiently solving computational science problems. In particular, the use of machine learning models is promising in problems where little is known about the underlying process, or in problems where the computational overhead is significant [42].

Data driven models. Sharifi et al. [32], Vandal et al. [40], Gentine et al. [14], and Srivastava et al. [36] use basic ML methods to interpolate between coarse and fine representations to improve accuracy

[1]The code for this paper is available at https://github.com/tech-submissions/physics-aware-downsampling

in modeling. In these studies, they use supervised data driven models that predict the solution of the physical model. Data driven models can fail to capture important complex relationships between physical variables directly from data, and it is necessary to ensure that the learned models follow physical laws.

Physics constrained loss function. To force the predictive models to follow physical laws, a common technique is to incorporate the physical constraints as additional terms in the loss function [19, 29, 30, 35, 4]. These physics-informed DNNs have several advantages. First, the computation of the physical loss term can be done without labeled data. In addition, the physical loss term acts as a regularizer, so it can reduce the search space of the optimization. These models has shown success in several tasks such as forward solving of PDEs and inverse problems [42]. However, physical loss terms require fine-tuning to find the optimal weighting between the predictive loss and the physical loss.

Physics guided architecture. Another common methodology is physics guided architecture, where domain knowledge is utilized to design architectures that capture physics dependencies among variables [33, 12, 9, 17, 24]. These models are more interpretable than regular models. However, the architecture design can be useful for a specific type of PDEs, and make no sense for others. Designing a specific architecture for each physical model can be a tedious task of its own.

Data driven discretization. Perhaps the most closely related works showed that modeling features of the fluid dynamics and numerical scheme with neural networks can enable solving PDEs on much coarser grids than is possible with standard numerical methods [4, 15, 21, 38]. By learning to modify the PDE solver, these works enable using a coarser grid (potentially with some overhead) while retaining accuracy and numerical stability. However, they did not examine how to optimally coarse grain the external environment (given as input to the solver) — as we do here, for terrain elevation maps in the context of flood modeling. In this case, it is critical that the coarse-grained map retains the relevant features, to prevent drastic differences between fine and coarse solutions (e.g., floods). In other words, these previous works address the discretization error entailed in numerical modeling, while this paper suggests a method to reduce the input error, such as errors in the elevation map. The two approaches (modifying the PDE solver vs. modifying the external environment) are orthogonal, and can be potentially combined in future work. Empirically, we found that the input error is dominant in our setting of flood modeling, however, it is an interesting question to quantify this more precisely. Note our approach has no computational overhead during inference since we use the same solver, just on a coarser grid with a modified map. Evidently, we provide experiments on a significantly larger scale than these previous works.

3 Physical Model: The Shallow Water Equations

In this section we describe the physical model of fluid flow used in this paper, i.e. the hydraulic model. At the most fundamental level, fluid flow is described by the Navier–Stokes equations. In practice, solving these equations directly is often infeasible, and simpler variations are derived for specific problem settings. One such variation is the shallow water equations [22], a set of partial differential equations that describe water flow when water depth is much smaller than horizontal extent, as is generally the case for floods. The shallow water equations are derived by depth-integrating the Navier–Stokes equations [41, 8, 7]. These assumptions allow a considerable simplification in the mathematical formulation and numerical solution, which enables analysis on much larger horizontal length scales.

To write the shallow water equations, we first define the following notation: t is the time index, (x, y) are the spatial horizontal coordinates, $h(x, y; t)$ is the water height relative to the terrain elevation $z(x, y)$, $\mathbf{q} = (q_x(x, y; t), q_y(x, y; t))$ is the horizontal mass flux of the fluid, $\nabla = (\frac{\partial}{\partial x}, \frac{\partial}{\partial y})$ is the horizontal gradient operator, n is Manning's friction coefficient, and g is the gravitational acceleration constant. The shallow water equations are described by two conservation laws. First, the conservation of mass, which gives rise to the continuity equation,

$$\frac{\partial h}{\partial t} + \nabla \cdot \mathbf{q} = 0\,. \tag{1}$$

Second, the conservation of momentum, which yields the momentum equation,

$$\frac{\partial \mathbf{q}}{\partial t} + gh\nabla\left(h + z\right) + \frac{gn^2\left|\left|\mathbf{q}\right|\right|}{h^{7/3}}\mathbf{q} = 0. \tag{2}$$

Note that eqs. 1 and 2 neglect advection and the Coriolis force, as is common for flood modeling. We follow De Almeida & Bates [7] to numerically approximate eqs. 1 and 2 from the continuous domain into the discrete domain, where the numerical solution uses a finite difference scheme applied to a staggered grid. In the discretized domain, functions becomes arrays (e.g., $z(x, y)$ becomes $\mathbf{z}$).

In real-world flood forecasting settings, data for the elevation map would be collected ahead of time, and the boundary conditions for the model would be provided either by real-time measurements or hydrologic forecasts. One would then run the hydraulic model (eqs. 1 and 2) in real-time, or offline on a wide range of possible boundary conditions. The simulation provides both the flood extent and the inundation depth at every pixel. This information can then be used to inform both authorities working on mitigation and response efforts, as well as individuals that may be affected by the expected flooding (e.g., alert apps on mobile phones). However, to do this, the simulation must be done on a large scale and fast enough to be practically relevant. Next, we show how this can be done using physics-aware downsampling of the elevation map.

4 Learning to Downsample the Elevation Map

In this section, we formulate the map downsampling problem for several application settings and define the training setup. Then, we describe data generation and configuration. Lastly, we describe the architecture considerations and optimization settings which enable the learning process success.

4.1 Problem Setup

Consider a downsampling DNN f_W and a numerical water flow solver S with fixed target time t. The downsampling network is a deep neural network parameterized by weights W. It transforms a two dimensional fine grid elevation map $\mathbf{z} \in \mathbb{R}^{m \times m}$ into a coarse grid elevation map $\hat{\mathbf{z}} \in \mathbb{R}^{k \times k}$ via the mapping $\hat{\mathbf{z}} = f_W(\mathbf{z})$. The fine and coarse grid water heights, $\mathbf{h} \in \mathbb{R}^{m \times m}$ and $\hat{\mathbf{h}} \in \mathbb{R}^{k \times k}$ respectively, are computed by the numerical solver S with initial and boundary conditions $\mathbf{c}$ such that

$$\mathbf{h} = S(\mathbf{z}, \mathbf{c}) \, , \, \hat{\mathbf{h}} = S(\hat{\mathbf{z}}, \mathbf{c}) \, .$$

A loss function $\ell(\mathbf{h}, \hat{\mathbf{h}})$ penalizes any difference between the fine grid water height $\mathbf{h}$ and the coarse grid water height $\hat{\mathbf{h}}$. The fine grid water height is downsampled to a coarse grid for calculation of the loss. Note that this differs from calculating the water height on a coarse grid, since the water height is calculated on a fine grid and only then is downsampled to a coarse grid. Since we are interested in inundation modeling in coarse resolution, the deviations in coarsening the fine water height are acceptable and can be accomplished by simple averaging of the fine water height. Figure 1 depicts the system overview. Our first general goal is to find a mapping f_W through the minimization problem:

$$W^* = \arg\min_W \mathbf{E}_{\mathbf{z},\mathbf{c}} \left[\ell \left(S(\mathbf{z}, \mathbf{c}), S(f_W(\mathbf{z}), \mathbf{c}) \right] . \tag{3}$$

The second goal we consider is a weaker variation of eq. (3), which is relevant in operational flood forecasting systems, as described in Sec. 5.2. In such a setting, the fine grid elevation map is known, and we wish to find a coarse representation of the elevation map that will generalize to many boundary conditions. We condition the expectation on $\mathbf{z}$ and the minimization problem can be rewritten as:

$$W^*(\mathbf{z}) = \arg\min_W \mathbf{E}_{\mathbf{c}} \left[\ell \left(S(\mathbf{z}, \mathbf{c}), S(f_W(\mathbf{z}), \mathbf{c}) \mid \mathbf{z} \right] . \tag{4}$$

To train and evaluate a model based on the minimization problem in eq. 3, training and validation data is required, which we next discuss how to configure.

4.2 Dataset configuration

Elevation map curation. In recent years, improvements in remote sensing and data processing at large scales have improved the access to reliable, diverse, and rich earth-science data. A case in point is the extensive collection of 1-meter Digital Elevation Models (DEMs) published by the United States Geological Survey (USGS) [1]. These DEMs are created using airborne lidar in 1-meter $\times$ 1-meter cell size resolution, and cover most of the United States, including flood-prone areas around the Mississippi River and the Arkansas River [13, 27]. We based our dataset on terrain data from the floodplains of those rivers in three different locations. These areas are a mere fraction of the available

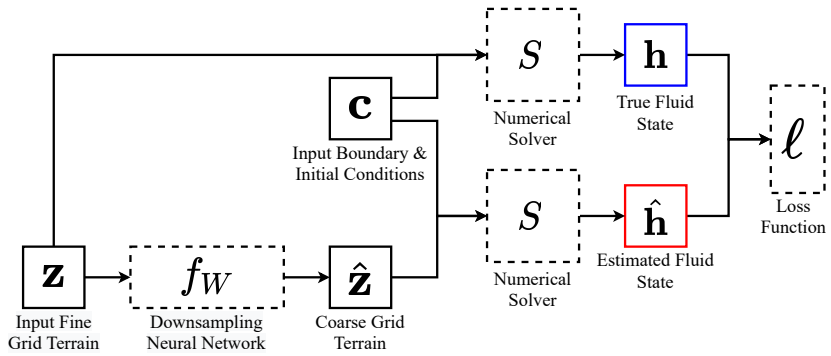

Figure 1: **Training setup for the downsampling neural network f_W.** Each data sample consists of a fine grid elevation map $\mathbf{z}$, with boundary and initial conditions $\mathbf{c}$. Two solutions are calculated - the fine grid water height $\mathbf{h}$ and the coarse grid water height $\hat{\mathbf{h}}$. The latter uses a coarse grid version of the input elevation map $\hat{\mathbf{z}}$. We train the downsampling neural network f_W, to minimize the loss measuring the distance between the two solutions (eqs. 3 or 4). This is done by backpropagating the loss gradients through the numerical solver.

USGS data, but we focused on them due to compute resource limitations. The data was further processed into tiled elevation maps. Mountainous terrains and areas with missing data were filtered to be appropriate for the hydraulic simulation. Each elevation map contains 2000×2000 pixels, covering a 2km×2km region. We found this size is large enough to capture the way differences between coarse and fine grid solutions can affect flooding even in distant locations. We gathered a total of 5183 elevation maps. For each elevation map, we defined boundary conditions as influx, outflux and discharge. More details about data configuration can be found in Sec. A.1 of the appendix.

Ground truth calculation. For the scope of this paper, we wish to calculate the flow of the water, i.e., the water height at each pixel, over each data sample, given its boundary conditions over some time t. It is possible to construct the corresponding ground truth by simulating the water flow over the fine grid elevation maps. For each elevation map, its ground truth is a 2000×2000 matrix, describing the water height at each pixel. This is a computationally demanding task, but it can be performed offline once. We provide access to the full dataset along with code to reproduce the data and expand to more data samples.

4.3 Architecture

We use the ResNet-18 architecture [16] as the core of our DNN model. The last layer of the classifier was removed, and a 1×1 convolution was used instead to transform the last hidden layer with multiple channels, into a single channel output. In our experiments, we use a downsampling factor of 16×, which is natively implemented in the ResNet-18 model. For a smaller downsampling factor it is possible to remove all layers with resolution lower than needed and apply the 1×1 convolution for a single channel output. It is also possible that other models can perform as well, but these kinds of experiments are beyond the scope of this paper.

For the output scheme, we first notice that simple average coarsening techniques works reasonably well for many of the elevation maps and for a large portion of the spatial domain. This is because fine important features effecting the hydraulic flow are relatively scarce. Therefore, we suggest that the downsampling DNN f_W output will consists a summation of an average pooling coarse grid elevation map and a correction map given by the ResNet-18 model. This is implemented by adding an average pooling skip connection from the input of the ResNet-18, directly to its output. This has two main advantages. First, the ResNet-18 output values have a smaller range and variance, in comparison to the full coarse grid elevation maps. Second, the starting point of the DNN accuracy is close to the accuracy of average pooling downsampling. Both advantages help the learning process converge significantly faster and make performance less sensitive to initialization — in comparison to the case where the DNN outputs the entire coarse grid elevation map rather than a correction (details in the Appendix).

Data normalization. It is common practice to normalize the input data of the DNN, e.g., by performing standardization, min-max normalization, and more. However, using multiplicative normalization such as standardization will distort the vertical scale of the elevation map and essentially remove important information. Indeed, two elevation maps can have the same normalized form, but encompass different hydraulic properties, and possibly different optimized coarse grid representation. Consequently, we normalize the input elevation map only by subtracting its mean.

4.4 Optimization scheme

The per-sample loss is a sum of the per-pixel loss ℓ on all the map pixels $\sum_{i,j} \ell(h_{i,j}, \hat{h}_{i,j})$. Here ℓ is Huber loss[2] which exhibited good convergence behavior. The training loss was optimized using Adam [20], and a batch size of 32 samples.

There are several challenges in training the DNN in the setting described in Sec. 4.1. First, to obtain gradients for each sample, one must calculate the hydraulic solution, i.e., iteratively applying the PDEs in the forward pass, and then calculate the gradients in the backward pass, through the PDEs. The backward calculation through the PDEs takes the majority of the iteration time. Ideally, we would like to reach a steady state solution, where the velocity field and total mass do not change. However, to keep the training computationally feasible, we stop the solver after it finished simulating one hour of hydraulic flow. This still might require thousands of recurrent iterations of the PDEs, depending on the CFL condition described in Sec. 3.

Gradient checkpointing. As discussed in Sec. 4.1, we define each fine grid elevation map to be relatively large, 2000×2000 pixels, to allow long-distance flow effects to manifest. As a result, the activation maps are large, causing a memory bottleneck and limiting the batch size that can be used in each node. In addition, the flow of water calculation might take thousands of recurrent iterations, as described in Sec. 1, increasing both memory and compute time. To address this computational challenge, we use the gradient checkpointing mechanism, as described in Siskind & Pearlmutter [34] and Chen et al. [6]. The checkpointing is applied both to the DNN, and to the numerical solver.

Numerical stability of the PDEs. In addition to instabilities originating in DNN non-convex optimization, one needs to take into account numerical instability of the PDEs when performing a learning rate scan. The numerical scheme of the Shallow Water Equations is prone to numerical instability issues, much like other numerical schemes. This instability is more apparent when there are large variations in the terrain, causing rapid changes in the water velocity and depth. When training with learning rates larger than 10^{-2} with Adam, and 10^{-1} with Stochastic Gradient Descent (SGD), divergence problems occur, especially at the beginning of training: over-correction by the gradients cause topographic exaggeration by the model which leads to numerical instability of the PDEs. To avoid this, we performed our learning rate scan with in the relatively low range of $[10^{-3}, 10^{-1}]$ to find the empirically optimal learning rate in terms of convergence rate and generalization, as well as numerical stability of the PDEs. Adam achieved better generalization and therefore is used.

5 Experiments

In this section, we leverage our physics-aware DNN to accurately solve the shallow water equations on a coarse grid, over a variety of elevation maps and boundary conditions. Specifically, we:

1. Demonstrate that the system (Sec. 4.1) can locate hydraulically significant details of the elevation map, using the accumulated gradient after backpropagation through the PDEs.
2. Optimize the DNN over a single elevation map with multiple boundary conditions (eq. 4) and demonstrate better performance compared to traditional downsampling. This is a typical setting in operational flood modeling systems, where repeated simulations are performed with different boundary conditions.
3. Train the DNN over a training set of different elevation maps and boundary conditions (eq. 3) and show generalization capabilities on both elevation maps and boundary conditions. This would enable fast downsamping of maps in new areas.

[2] i.e. $\ell(h, \hat{h}) = 0.5(h - \hat{h})^2$ if $|h - \hat{h}| \leq 1$ and $\ell(h, \hat{h}) = \left|h - \hat{h}\right| - 0.5$ otherwise.

Throughout the experiments, we use an averaging baseline for comparison. This baseline is intuitive for terrain elevation map downsampling, and is being used in traditional inundation modeling studies [11, 43]. We also consider slightly less naive downsampling techniques, and we demonstrate that they are not superior to the baseline we use (Appendix A.2.3). More details on the experiments can be found in Appendix A.2.

5.1 Backpropagation through PDEs

In this section, we demonstrate that hydraulically meaningful details in the elevation map are identified by backpropagation through the PDEs, allowing the model to learn to preserve these details when downsampling. Specifically, we show that the gradient signal can properly flow even through *tens of thousands* of unrolled recurrent units representing the PDE time-steps. We explicitly construct an example to demonstrate that important details in the elevation map can be detected by the gradients: We start with an elevation map with an evident embankment, and artificially flatten a few pixels of that embankment. The modified elevation map is shown in Fig. 2a with a red square around the flattened pixels (a zoomed-in version of the red square is shown in Fig. 2b-top). From the boundary conditions we used, water flows in from the right bottom part of the elevation map and flows out from right upper part. With the flattened pixels, water can flow through the embankment, inundating a large area left of the embankment which is not inundated in the non-modified elevation map. This difference is shown in Fig. 2d, where red pixels are more inundated on the modified elevation map solution (over-flooding), and blue pixels are more inundated on the non-modified elevation map solution (under-flooding). In turn, the loss is high because of the large inundated area. A meaningful gradient will identify the discrepancy as resulting from the flattened embankment pixels, rather than the entire inundated region. The resulting gradient map of the elevation map after backpropagation is shown in Fig. 2c. As can be seen, the gradient is highest at the flattened pixels, especially compared to the inundated region, indicating that those few pixels contributed significantly to the deviation from the non-modified elevation map solution. We emphasize that it was required to unroll more than ten thousand iterations to compute the gradient, as we simulated 6 hours of hydraulic flow.

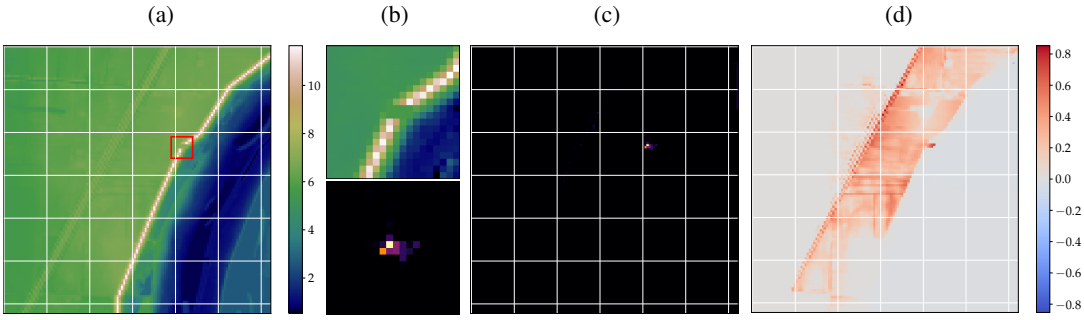

Figure 2: **Gradient flow through PDEs.** (**a**) A modified elevation map with a few pixels of the embankment flattened . (**d**) Difference map between the solution calculated on the modified elevation map and the non-modified elevation map solution. (**c**) Gradient magnitude of the modified elevation map, computed by back-propagating through the PDEs. (**b**) Zoom-in on the modified pixels of the elevation map (top) and on the gradient magnitude map of the same location (bottom).

5.2 Generalization over multiple boundary conditions

In this section, we find an optimized coarse grid representation of a single elevation map with varying boundary conditions, as in the minimization problem in Eq. (4). To do so, we optimize a DNN where each data sample uses the same elevation map, but with different boundary conditions — including influx and outflux locations as well as discharge. This setting is useful for repeated simulations with different boundary conditions over the same spatial domain, a typical setting in operational flood modeling systems [5]. We validate the model on randomly sampled boundary conditions. The baseline we compare ourselves against remains average pooling. In Figure 3 we show a comparison between the water flow solution on a fine grid elevation map and each of the solutions calculated on coarse grid elevation maps, both for the trained DNN and baseline. As can be seen, solutions calculated on the DNN coarse grid elevation map are closer to the fine grid solution. We provide more examples on different elevation maps in Appendix A.2.4.

In Figure 4 we provide example of visible features preserved by the DNN. On the right image, there is a small embankment that is preserved entirely in the DNN version, while in the baseline version the embankment is averaged with its surroundings, causing it to almost disappear. On the left image, there is a narrow canal with embankments on either side. Again, the DNN preserves both the canal and most of the embankments, while in the baseline version, the canal is averaged with the embankments, causing partial flattening of the terrain at the canal location. We emphasize that the DNN coarse grid elevation maps are outputs of two different models that were trained on each image separately.

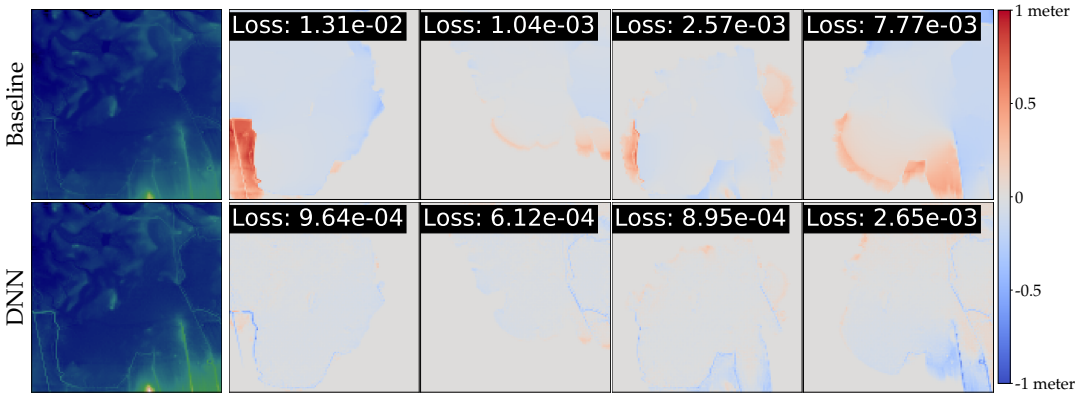

Figure 3: **Generalization over different boundary conditions.** Leftmost images are coarse grid elevation maps downsampled using a DNN (bottom) and average pooling baseline (top), where the DNN trained on different boundary conditions and the same elevation map. Each column is a comparison of solutions between the DNN and the baseline elevation maps, for different boundary conditions. Each image on those columns is a difference map between the coarse and fine grid solutions along with the Huber loss. Red indicates pixels more inundated in the coarse grid solution, and blue indicates pixels more inundated in the fine grid solution. The color scale, in meters, appears on the right. Solutions calculated on the DNN elevation map achieve better accuracy compared to the baseline.

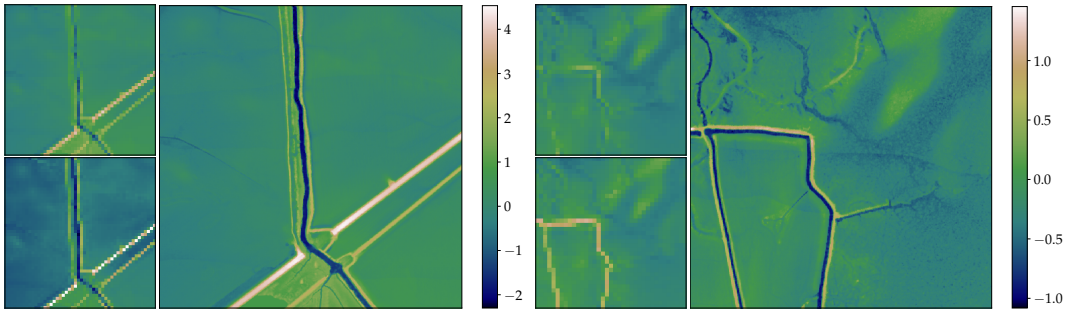

Figure 4: **Capturing fine details important to water flow.** Two patches taken from different elevation maps. The two larger images are the fine grid patches and the two smaller images on the left of each fine grid image are the coarse grid patches of the average pooling baseline (top) and the DNN (bottom). Note the preservation of hydraulically significant features such as canals and embankments in the DNN as opposed to the baseline.

5.3 Generalization over different elevation maps

For scalable inundation modeling, a robust model for coarse grid representation is required, which is able to generalize to elevation maps and boundary conditions unseen during training. We train such model on 4031 elevation maps, each with different boundary conditions. We validate the model on 1155 unseen elevation maps, each with different boundary conditions. We compare the accuracy of the water flow solutions obtained on a coarse grid between the DNN elevation maps and the average pooling elevation maps. The metric used for water flow solution accuracy is Huber loss. We used Adam with a learning rate 10^{-3}, and a batch size of 32 samples. The training was done in 90 epochs. Figure 5a shows the convergence plot of the model and its improvement over the baseline. Note that

the model starts with a loss similar to the baseline, because of the architecture scheme described in section 4.3.

Next, we plot the 2D histogram of the DNN's loss compared to the baseline (Figure 5b). The resulting distribution is almost entirely below the linear dashed line, indicating better accuracy by the DNN, matching Figure 5b. There is a large number of data points with small loss (both by the model and the DNN), meaning that for a large portion of the data, average pooling achieves reasonable accuracy. Overall, the modified elevation maps exhibited an improvement of the Huber loss by a factor of 2 over all the tested elevation maps. This improvement increases to a factor of 2.5 as we look only at harder examples (90% percentile). Other accuracy metrics show similar results with a larger improvement factor of up to 3. Additional analysis is provided in Appendix A.2.5.

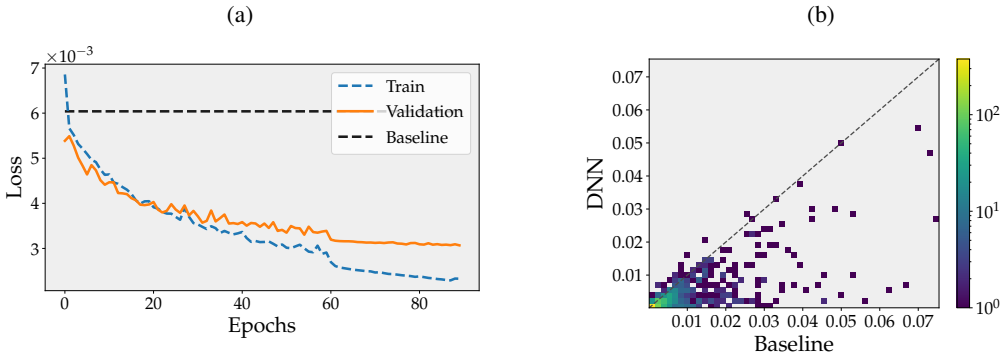

Figure 5: **Generalization over different elevation maps.** (**a**) - Convergence plot of the model loss for both training and validation. (**b**) - A comparison of the per-sample Huber loss values of the baseline (x-axis) and the DNN (y-axis). The color scale is logarithmic. The DNN model surpass the accuracy of the baseline, with an improvement of the Huber loss by a factor of 2. Additional analysis is provided in Appendix A.2.5

6 Summary

Enabling accurate inundation modeling is a vitally important endeavor for flood forecasting, and physical water flow simulations are typically a core component of that endeavor. These simulations rely on accurate elevation maps, and computationally demanding on a large scale. Elevation map coarsening can significantly reduce the computation required, but hydraulically important fine details might get distorted on a coarse grid. We tackle this problem by training a physics aware DNN that downsamples elevation maps to a coarse grid representation, while preserving details important for hydraulic simulation. We do so by incorporating a physical model into the learning process, such that gradients are calculated through the PDEs. We also configure a dataset specifically for this task and make it available for further use. We consider two applicable training settings for operational flood forecasting systems, and demonstrate that such training is feasible and that the learned DNN outputs meaningful elevation maps with better water flow results compared to prevalent methods.

7 Broader impact

Floods cause thousands of fatalities, affect hundreds of millions of people, and cost approximately $10 billion every year [10, 39]. Flood early warning systems have the potential to significantly reduce both fatalities and economic costs, in many cases between 30-50% [31, 25, 2, 28]. Inundation models are a critical component of such systems, as many people will not take protective action without information that is spatially accurate [5]. However, the vast majority of fatalities and harms occur in low and middle income countries, where both financial resources and expertise to support such efforts is scarce [10, 26]. As a result, the requirement of enormous computational resources, or the expertise to intelligently downscale elevation maps to enable more efficient simulation, can be infeasible for the most important uses of these models. This work can bypass the need for either of these constraints, providing a tool for the resource-constrained to use in their modeling, or helping support efforts to scale up global inundation models, such as those pursued by organized by Google, Fathom, ECMWF, CGIAR and others.

8 Acknowledgments

We thank Elad Hoffer for technical advising, as well as Yaniv Blumenfeld and Mor Shpigel Nacson for valuable comments on the manuscript. The research of DS was supported by the Israel Science Foundation (grant No. 1308/18), and by the Israel Innovation Authority (the Avatar Consortium).

References

[1] S.T. Arundel, C.M. Archuleta, L.A. Phillips, B.L. Roche, and E.W. Constance. 1-meter digital elevation model specification: U.S. Geological Survey Techniques and Methods, 2015.

[2] World Meteorological Association et al. The global climate 2001–2010: A decade of climate extremes, summary report. *Geneva, Switzerland: WMO, 16p*, 2013.

[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[4] Yohai Bar-Sinai, Stephan Hoyer, Jason Hickey, and Michael P. Brenner. Learning data driven discretizations for partial differential equations. *Proceedings of the National Academy of Sciences of the United States of America*, 116, 2019.

[5] Zvika Ben-Haim, Vladimir Anisimov, Aaron Yonas, Varun Gulshan, Yusef Shafi, Stephan Hoyer, and Sella Nevo. Inundation Modeling in Data Scarce Regions. 2019.

[6] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016.

[7] Gustavo A.M. De Almeida and Paul Bates. Applicability of the local inertial approximation of the shallow water equations to flood modeling. *Water Resources Research*, 49, 2013.

[8] Gustavo AM de Almeida, Paul Bates, Jim E Freer, and Maxime Souvignet. Improving the stability of a simple formulation of the shallow water equations for 2-d flood modeling. *Water Resources Research*, 48(5), 2012.

[9] Emmanuel De Bézenac, Arthur Pajot, and Patrick Gallinari. Deep learning for physical processes: Incorporating prior scientific knowledge. In *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, 2018.

[10] Shannon Doocy, Amy Daniels, Catherine Packer, Anna Dick, and Thomas D Kirsch. The human impact of earthquakes: a historical review of events 1980-2009 and systematic literature review. *PLoS currents*, 5, 2013.

[11] TJ Fewtrell, Paul David Bates, Matt Horritt, and NM Hunter. Evaluating the effect of scale in flood inundation modelling in urban environments. *Hydrological Processes: An International Journal*, 22(26):5107–5118, 2008.

[12] Stathi Fotiadis, Eduardo Pignatelli, Anil A Bharath, Mario Lino Valencia, Chris D Cantwell, and Amos Storkey. ICLR 2020 Workshop on Deep Learning and Differential Equations COMPARING RECURRENT AND CONVOLUTIONAL NEU-RAL NETWORKS FOR PREDICTING WAVE PROPAGATION. Technical report.

[13] Gerry Galloway. Usa: flood management–mississippi river. *WMO/GWP Associated Programme on Flood Management*, 2004.

[14] Pierre Gentine, Mike Pritchard, Stephan Rasp, Gael Reinaudi, and Galen Yacalis. Could machine learning break the convection parameterization deadlock? *Geophysical Research Letters*, 45(11):5742–5751, 2018.

[15] Daniel Greenfeld, Meirav Galun, Ron Kimmel, Irad Yavneh, and Ronen Basri. Learning to optimize multigrid PDE solvers. In *36th International Conference on Machine Learning, ICML 2019*, 2019.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[17] Philipp Holl, Vladlen Koltun, and Nils Thuerey. Learning to Control PDEs with Differentiable Physics. In *8th International Conference on Learning Representations, ICLR 2020*, 2020.

[18] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456. PMLR, 2015.

[19] Anuj Karpatne, Gowtham Atluri, James H Faghmous, Michael Steinbach, Arindam Banerjee, Auroop Ganguly, Shashi Shekhar, Nagiza Samatova, and Vipin Kumar. Theory-guided data science: A new paradigm for scientific discovery from data. *IEEE Transactions on knowledge and data engineering*, 29(10):2318–2331, 2017.

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

[21] Dmitrii Kochkov, Jamie A Smith, Ayya Alieva, Qing Wang, Michael P Brenner, and Stephan Hoyer. Machine learning accelerated computational fluid dynamics. *arXiv preprint arXiv:2102.01010*, 2021.

[22] Pijush K Kundu and Ira M Cohen. Fluid mechanics. 2002.

[23] Randall J LeVeque and Randall J Leveque. *Numerical methods for conservation laws*, volume 132. Springer, 1992.

[24] Zichao Long, Yiping Lu, Xianzhong Ma, and Bin Dong. Pde-net: Learning pdes from data. In *International Conference on Machine Learning*, pp. 3208–3216. PMLR, 2018.

[25] World Health Organization et al. *Global report on drowning: preventing a leading killer*. World Health Organization, 2014.

[26] P Peduzzi, U Deichmann, A Maskrey, FA Nadim, H Dao, B Chatenoux, C Herold, A Debono, G Giuliani, and S Kluser. Global disaster risk: patterns, trends and drivers. *Global assessment report on disaster risk reduction*, pp. 17–57, 2009.

[27] Charles A Perry. *Significant floods in the United States during the 20th century: USGS measures a century of floods*. US Department of the Interior, US Geological Survey, 2000.

[28] Paul J Pilon. Guidelines for reducing flood losses. Technical report, United Nations International Strategy for Disaster Reduction (UNISDR), 2002.

[29] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations. *arXiv preprint arXiv:1711.10561*, 2017.

[30] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

[31] David Rogers and Vladimir Tsirkunov. Costs and benefits of early warning systems. *Global assessment rep*, 2011.

[32] Ehsan Sharifi, B Saghafian, and R Steinacker. Downscaling satellite precipitation estimates with multiple linear regression, artificial neural networks, and spline interpolation techniques. *Journal of Geophysical Research: Atmospheres*, 124(2):789–805, 2019.

[33] Justin Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375:1339–1364, 2018.

[34] Jeffrey Mark Siskind and Barak A Pearlmutter. Divide-and-conquer checkpointing for arbitrary programs with no user annotation. *Optimization Methods and Software*, 33(4-6):1288–1330, 2018.

[35] Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33, 2020.

[36] Prashant K Srivastava, Dawei Han, Miguel Rico Ramirez, and Tanvir Islam. Machine learning techniques for downscaling smos satellite soil moisture using modis land surface temperature for hydrological application. *Water resources management*, 27(8):3127–3144, 2013.

[37] Carlo Tomasi and Roberto Manduchi. Bilateral filtering for gray and color images. In *Sixth international conference on computer vision (IEEE Cat. No. 98CH36271)*, pp. 839–846. IEEE, 1998.

[38] Kiwon Um, Robert Brand, Philipp Holl, Nils Thuerey, et al. Solver-in-the-loop: Learning from differentiable physics to interact with iterative pde-solvers. *arXiv preprint arXiv:2007.00016*, 2020.

[39] CRED UNISDR et al. The human cost of natural disasters: A global perspective. 2015.

[40] Thomas Vandal, Evan Kodra, Sangram Ganguly, Andrew Michaelis, Ramakrishna Nemani, and Auroop R Ganguly. Deepsd: Generating high resolution climate change projections through single image super-resolution. In *Proceedings of the 23rd acm sigkdd international conference on knowledge discovery and data mining*, pp. 1663–1672, 2017.

[41] Cornelis Boudewijn Vreugdenhil. *Numerical methods for shallow-water flow*, volume 13. Springer Science & Business Media, 1994.

[42] Jared Willard, Xiaowei Jia, Shaoming Xu, Michael Steinbach, and Vipin Kumar. Integrating physics-based modeling with machine learning: A survey. *arXiv preprint arXiv:2003.04919*, 2020.

[43] Oliver E. J. Wing, Paul D. Bates, Jeffrey C. Neal, Christopher C. Sampson, Andrew M. Smith, Niall Quinn, Iuliia Shustikova, Alessio Domeneghetti, Daniel W. Gilles, Radoslaw Goska, and Witold F. Krajewski. A New Automated Method for Improved Flood Defense Representation in Large-Scale Hydraulic Models. *Water Resources Research*, 2019.

[44] Dapeng Yu and Stuart N Lane. Urban fluvial flood modelling using a two-dimensional diffusion-wave treatment, part 1: mesh resolution effects. *Hydrological Processes: An International Journal*, 20(7):1541–1565, 2006.

A Supplementary Material

A.1 Data Configuration

The inputs to a hydraulic simulation include an elevation map, initial conditions, and the boundary conditions. For a given elevation map, there is an infinite possible combinations of initial and boundary conditions that could potentially realize in future events. It is an interesting question how to automatically configure the most relevant initial and boundary conditions to train on, to get a representation that will be useful in potential future real-world scenarios. We suggest a basic configuration that adequate for the purpose of this paper.

Initial conditions. These include the water height $\mathbf{h} \in \mathbb{R}^{m \times m}$ at each pixel and a staggered grid flux $\mathbf{q} \in \mathbb{R}^{2 \times (m-1) \times (m-1)}$ in each direction x, y. For simplicity, since we are interested in the steady state solution so the initial conditions is less relevant, we used zero initial conditions, meaning no water at the beginning of the simulation. We note that it is possible to obtain physically reasonable initial conditions by taking snapshots of $\mathbf{h}$ and $\mathbf{q}$ along a simulation and use it as initial conditions.

Boundary conditions. These include the locations and widths where water flows in and out along with the discharge of the water that flows in (influx) and the slope of the water that flows out (outflux). It is not obvious how to automatically define reasonable realistic boundary conditions. For example, for a single river crossing the elevation map, it would make sense to define the influx at the upstream edge of the river and accordingly at the downstream edge of the river. It gets more complicated as river branches appear, and even more complicated where no river exists at all, as happens in many of the elevation maps. We suggest a basic heuristic approach to automatically define the boundary conditions. First, we find the lowest point on the boundary of the elevation map, and define this point as the outflux location. Then, the influx location is defined to be the middle of the opposite axis of the chosen outflux axis. We define it this way to get a large flood extent, by constraining the water to cross the elevation map before it can flow out. We define both the influx and outflux widths to be 400 meters. The data configured with this method is used in Section 5.3. In Section 5.2, a different method is used where we define multiple boundary conditions for the same elevation map, as described in Section A.2.4.

As described in Section 3, we follow De Almeida & Bates [7] to discretize the shallow water equations. The discrete implementation described in De Almeida & Bates [7] uses two parameters – a weighting factor θ that adjusts the amount of artificial diffusion, and a coefficient $0 < \alpha \leq 1$ that is used as a factor by which we multiply the time step. We use the proposed values in De Almeida & Bates [7], namely $\theta = 0.7, \alpha = 0.7$.

A.2 Experiments Details

A.2.1 Technical details

The experiments were conducted on two configurations of machines. For experiments described in Section 5.2, we used distributed data parallelism on a machine with 4 Nvidia RTX2080TI GPUs. For the experiments described in Section 5.3 we used distributed data parallelism on a machine with 8 Nvidia RTX2080TI GPUs.

Handling a varying number of iterations. At each iteration t, the time step $dt^{(t)}$ depends on the current maximum water height, $\max_{i,j} h_{i,j}^{(t)}$, following the CFL condition described in Section 3. For different boundary conditions and elevation maps, $\max_{i,j} h_{i,j}^{(t)}$ can vary, yielding a different time step for each iteration t, and ultimately a different number of iterations until the required time of water flow simulation is achieved. In batch processing, each sample in the batch finishes the forward pass calculation in a different number of iterations. To resolve this issue, we perform iterations with $dt^{(t)} = 0$ for samples that finished their calculation until all samples in the batch are done.

A.2.2 Choosing the loss function

For the optimization process we examined two loss functions: the mean squared error (MSE)

$$\ell_{\text{MSE}}(h, \hat{h}) = (h - \hat{h})^2$$

and the Huber loss (also referred as smooth ℓ_1)

$$\ell_{\text{Huber}}(h, \hat{h}) = \begin{cases} 0.5(h - \hat{h})^2 & \text{, if } |h - \hat{h}| \leq 1 \\ \left|h - \hat{h}\right| - 0.5 & \text{, if } |h - \hat{h}| > 1 \end{cases}$$

summing the loss over samples and pixels. In addition, for test purposes we defined a new heuristic loss which aims to capture meaningful floods for inundation modeling purposes (similarly to how the 0-1 loss captures meaningful errors in classification problems). This new loss, called "Inundation Error", equals 1 if the difference between the predicted and ground true water height is above a certain threshold parameter c, and 0 otherwise:

$$\ell_{\text{Inundation}}(h, \hat{h}; c) = \begin{cases} 1, & \text{for } |h - \hat{h}| \geq c \\ 0, & \text{otherwise} \end{cases}$$

The motivation for this new loss is based on the fact that, in flood forecasting warning systems, areas inundated above some water height must be evacuated, regardless of the exact water height. Therefore, in this setting, errors larger than some predetermined water height level have the same meaning for inundation modeling. A value $c = 0.5$ meter is used for the inundation error, as a real threshold estimate for flood forecasting systems.

Fig. A.1 shows a comparison between the three loss functions, when using the baseline average pooling. As can be seen, the loss values of all loss functions are highly correlated. Therefore, the training can potentially be done with either of the first two differentiable loss functions. However, the correlation of the inundation error with the Huber loss is somewhat higher than the MSE. Also, the Huber loss exhibited better convergence properties. Therefore, it was used for training.

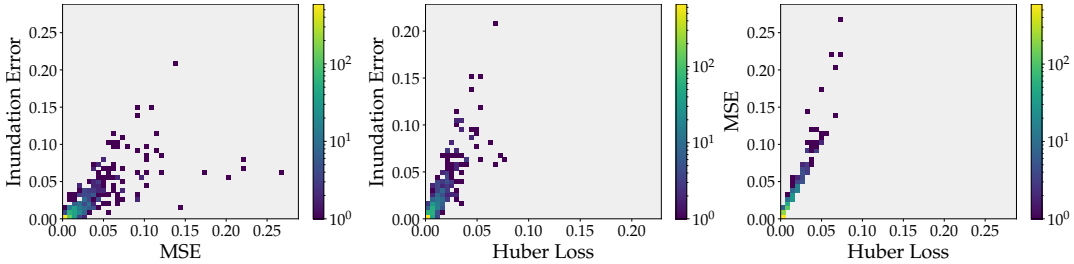

Figure A.1: **Correlation between loss metrics.** A comparison of the per-sample loss values for average pooling with different loss functions. The colorscale is logarithmic. The corresponding Pearson correlations from left to right are 0.78, 0.88, and 0.97

A.2.3 Edge preserving downsampling

It is reasonable to ask whether there exist downsampling techniques that perform better than average pooling, but do not require the heavy machinery of deep learning. In this experiment, we show two slightly less naive downsampling techniques and demonstrate that they do not successfully capture the intricacies of hydraulically meaningful downsampling. Specifically, we use a bilateral filter [37] to smooth the elevation map while preserving edges, before downsampling. We then downsample the map using either average pooling or max-pooling. The rationale behind max-pooling is that we seek to preserve high-elevation features such as embankments, whose height is reduced by average pooling.

However, as shown in Fig. A.2 and A.3, neither of these techniques outperforms the naive average pooling method used in the main text. This reinforces the claim that machine learning is an appropriate tool for hydraulically-aware downsampling.

A.2.4 Generalization over multiple boundary conditions

In this section, we provide more details of the experiment depicted in Section 5.2. The DNN was trained on a training set of 96 samples with SGD with momentum, learning rate 0.01, and the Huber loss. Recall that each sample uses the same map, but with different boundary conditions. We used 32 equally spaced influx locations along the boundaries of the elevation map. The outflux locations were

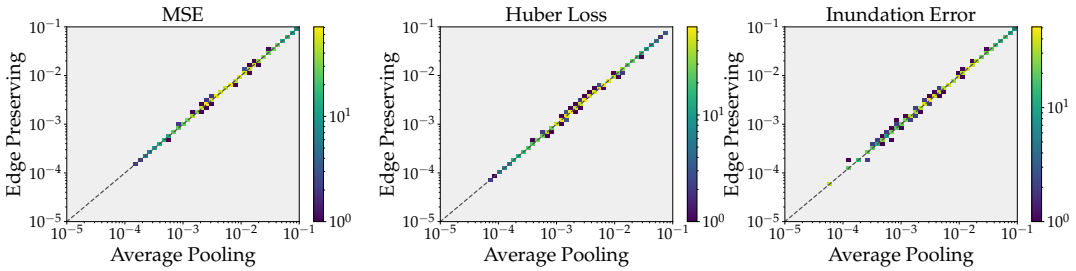

Figure A.2: A comparison of the per-sample loss values of average pooling (x-axis) and edge preserving smoothing followed by average pooling (y-axis). The colorscale is logarithmic. The two methods have a similar per-sample loss values.

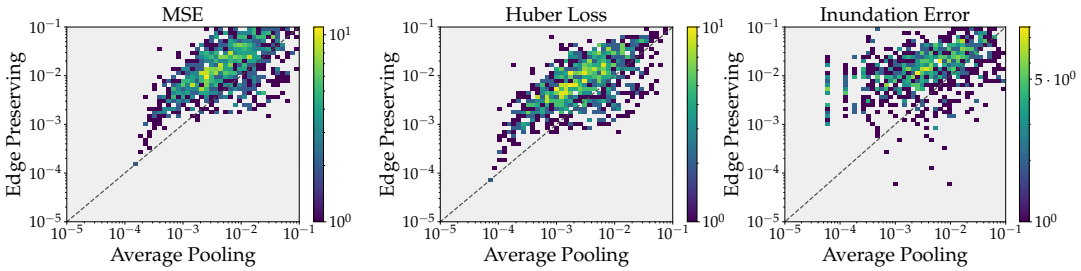

Figure A.3: A comparison of the per-sample loss values of average pooling (x-axis) and edge preserving smoothing followed by max pooling (y-axis). The colorscale is logarithmic. Average pooling typically has a lower loss.

defined in the middle of the edge opposite that of the influx. Each location was experimented with 3 different discharges, for a total of 96 samples. The test set is 100 boundary conditions randomly sampled along the boundaries of the elevation map with a random discharge for each boundary condition. We repeated the experiment in Section 5.2 with different elevation maps randomly selected. Results are shown in Fig. A.5, A.6, A.7, and A.8. A comparison of the per-sample loss values between the baseline and the DNN is provided for each of these examples are shown in Fig. A.4. As can be seen, a DNN trained on a single elevation map with multiple boundary conditions is able generalize well to other boundary conditions. In addition, there are examples where the DNN is not significantly different from the baseline. This can happen when the baseline already works quite well (e.g., as in Fig. A.8).

A.2.5 Generalization over different elevation maps

We provide more details of the experiment described in Section 5.3. In Table A.1, we provide results which correspond to Fig. 5, along with comparison for different loss functions. This comparison emphasizes that the DNN obtains more significant improvement compared to the baseline for the "hard samples", i.e., the samples with the largest baseline error. This implies that the DNN downsampling is most effective for elevation maps with fine important details that the baseline fails to preserve accurately on a coarse grid.

In addition, we provide a few examples of elevation maps taken from the test set for the DNN trained in Section 5.3. Fig. A.10, A.11, and A.12 shows these examples where we compare the baseline, the DNN, and the DNN correction to the baseline (the ResNet-18 output). As can be seen, the DNN does non-trivial corrections to the map, to preserve hydraulically important details.

A.2.6 Generalization over long times and geographical regions

In this experiment, we wish to examine the impact of the DNN corrections over long time of hydraulic flow. In addition we examine whether our training method described in Section 5.2 performs well on a different geographical region. We train a DNN over an elevation map of the Ganges river, west of Patna, India, where floods are a major concern every year, during the monsoon season. The elevation map has a resolution of 4 meters $\times$ 4 meters per pixel, and contains 2800$\times$2800 pixels, covering an

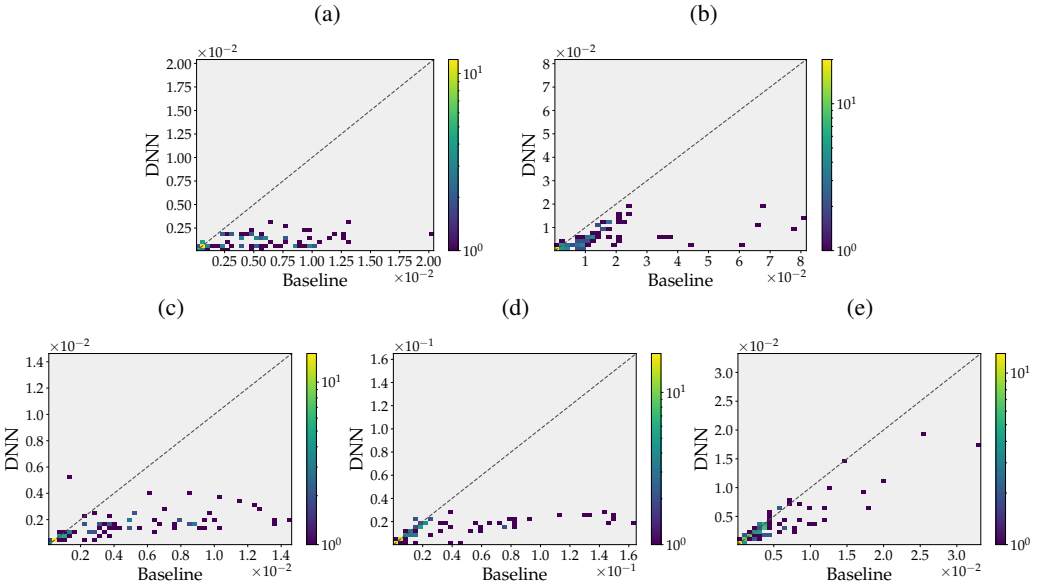

Figure A.4: Huber loss per-sample comparison of the baseline (x-axis) and the DNN (y-axis). Each panel is a DNN trained on a different elevation map. The colorscale is logarithmic. the panels correspond to the following figures – (**a**) Fig. 3 (**b**) Fig. A.5 (**c**) Fig. A.6 (**d**) Fig. A.7 (**e**) Fig. A.8

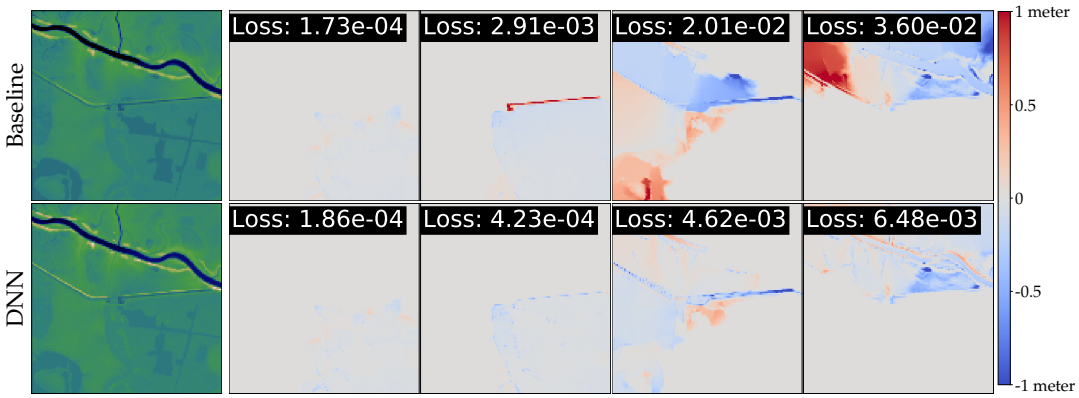

Figure A.5: **Generalization over different boundary conditions.** See Fig. 3 for details.

11.2km×11.2km region. The DNN is trained with the same influx and outflux locations as specified in Sections 5.2 and A.2.4, and with larger discharges that correspond to the much larger Ganges river. Each sample in the training set represents 1 hour of hydraulic simulation. We use the trained DNN to downsample the elevation map into a 175×175 coarse grid map with a resolution of 64 meters × 64 meters per pixel. We then simulate 20 hours of hydraulic flow from two influx locations at the upstream edge of the river and one outflux location at the downstream edge of the river. Results are shown in Fig. A.9. As can be seen, there is a large region on the left side of the map which is inundated only in the baseline solution. In this region, a long, narrow embankment protects a number of villages from flooding; the DNN maintains the correct height of the embankment, but the baseline solution averages out the embankment height with its surroundings, causing substantial overflooding. In addition, a river branch at the bottom right side is (correctly) inundated only in the DNN solution. This area remains dry in the baseline solution because the naive coarsening incorrectly flattens out a narrow channel leading to the river branch. This experiment demonstrates that training over relatively short periods of time is sufficient to significantly improve the accuracy, even when simulating hydraulic flow on a coarse grid for long periods of time. In addition, the training regime of Section A.2.4 seems to generalizes well to a completely different region from which we tuned it on.

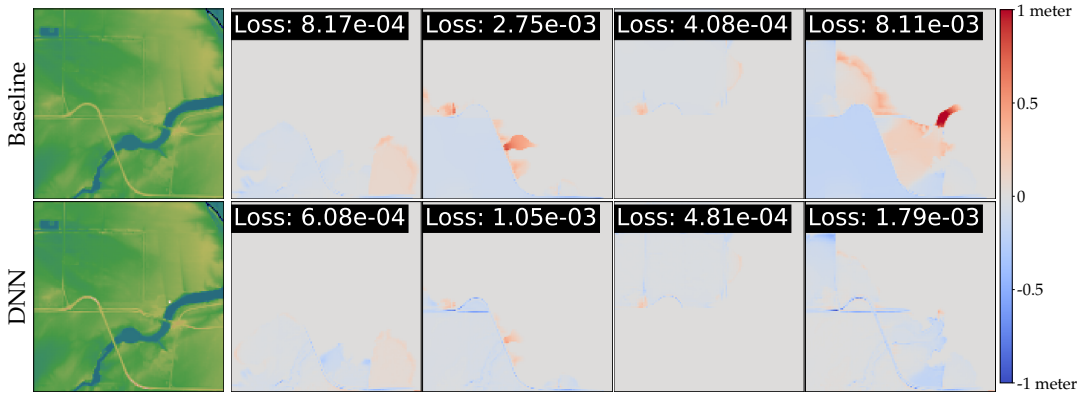

Figure A.6: **Generalization over different boundary conditions.** See Fig. 3 for details.

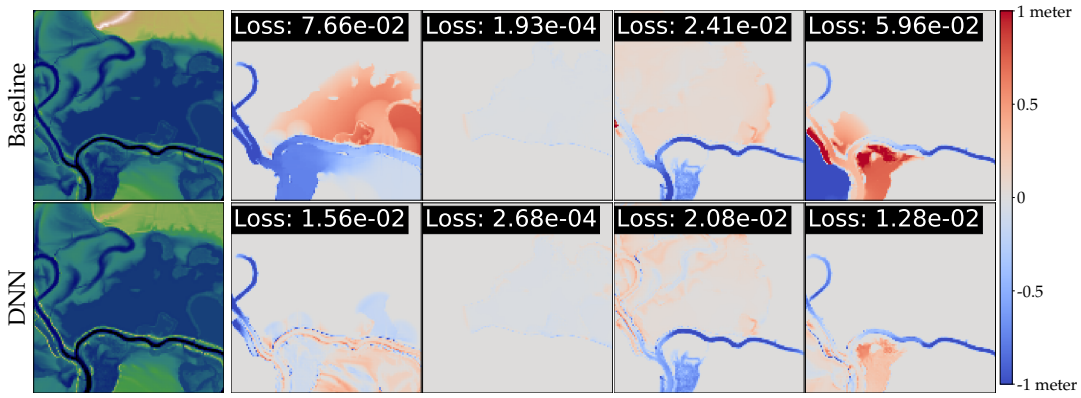

Figure A.7: **Generalization over different boundary conditions.** See Fig. 3 for details.

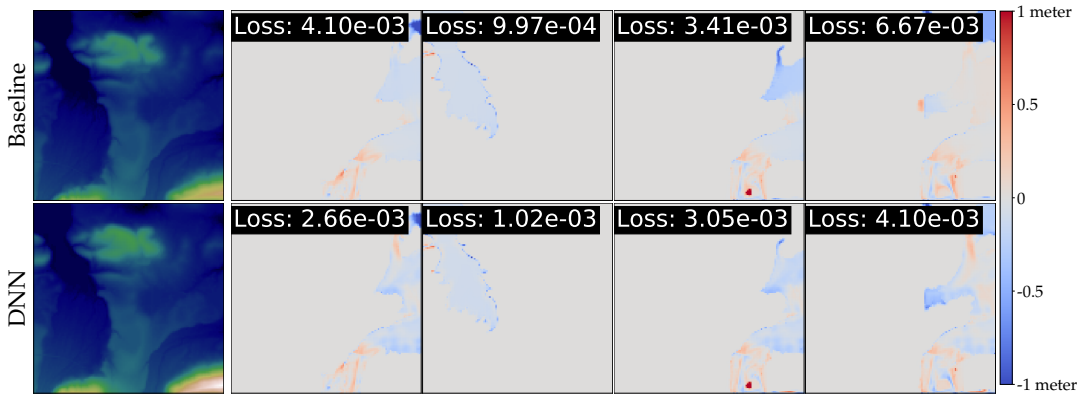

Figure A.8: **Generalization over different boundary conditions.** See Fig. 3 for details.

A.3 Architecture considerations

We first considered a plain ResNet-18 model with its classifier removed as the downsampling DNN. However, this DNN fails to capture the elevation scale information of the elevation map, meaning that the DNN outputs a coarse grid elevation map with different elevation scale. This failure occurs due to the normalization layers applied in ResNet-18. Normalization layers such as batch normalization [18] or layer normalization [3] are commonly used in modern DNNs, and it has been shown numerous

Table A.1: **Generalization over different elevation maps.** A comparison between average pooling and the DNN trained in Section 5.3 for different loss functions and percentiles. Notably, on samples with a large error, the DNN obtains more significant improvement (compared to the baseline).

Loss	Average pooling	DNN	Error ratio
Huber loss	$6.04 \cdot 10^{-3}$	$3.07 \cdot 10^{-3}$	1.97
Huber loss 50 percentile	$1.09 \cdot 10^{-2}$	$5.11 \cdot 10^{-3}$	2.13
Huber loss 90 percentile	$2.80 \cdot 10^{-2}$	$1.10 \cdot 10^{-2}$	2.55
MSE	$1.37 \cdot 10^{-2}$	$6.64 \cdot 10^{-3}$	2.06
MSE 50 percentile	$2.44 \cdot 10^{-2}$	$1.12 \cdot 10^{-2}$	2.18
MSE 90 percentile	$6.63 \cdot 10^{-2}$	$2.69 \cdot 10^{-2}$	2.92
Inundation error	$9.77 \cdot 10^{-3}$	$4.44 \cdot 10^{-3}$	2.20
Inundation error 50 percentile	$1.89 \cdot 10^{-2}$	$8.14 \cdot 10^{-3}$	2.32
Inundation error 90 percentile	$5.64 \cdot 10^{-2}$	$1.77 \cdot 10^{-2}$	3.19

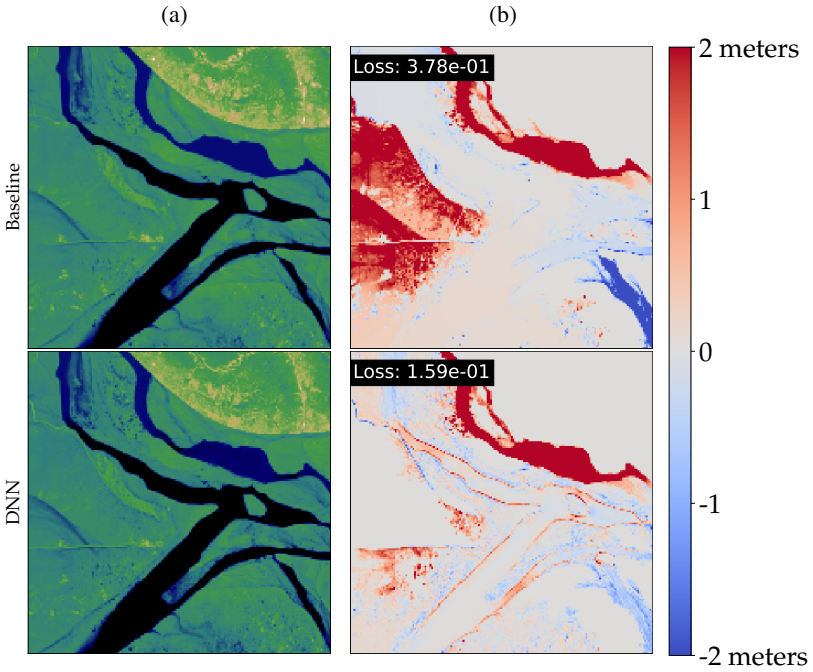

Figure A.9: **20 hours of hydraulic flow over a different geographical region.** A comparison of the solution obtained on a coarse grid between the baseline and a DNN trained on samples of 1 hour of hydraulic flow. Water flows in from bottom left and top left edges of the river, and flows out for the right edge of the river. (**a**) The coarse grid elevation map of the baseline (top) and the DNN (bottom). (**b**) Difference map between the coarse and fine grid solutions along with the Huber loss. Red indicates pixels more inundated in the coarse grid solution, and blue indicates pixels more inundated in the fine grid solution. The solution calculated on the DNN elevation map achieve better accuracy although the DNN was trained on much shorter time periods and a completely different region from which we tuned the training regime.

times that using normalization layers can improve the optimization and generalization of neural networks. However, for the DNN to extract features of the elevation map in different layers, the elevation scale information of the input has to be propagated. Using normalization layers prevents this elevation scale signal propagation.

To resolve this problem, we removed the normalization layers only in the residual connection to enable signal propagation without scale distortion. Indeed, the elevation scale is preserved with this architecture modification, and better convergence and generalization is obtained. Still, the DNN described in the paper, where a skip connection with average pooling is applied from the ResNet-18 input directly to its output, achieves better performance (achieving better initial performance and

faster convergence rate), compared to the modified ResNet-18 architecture. Therefore, we opted to use the architecture from the main paper.

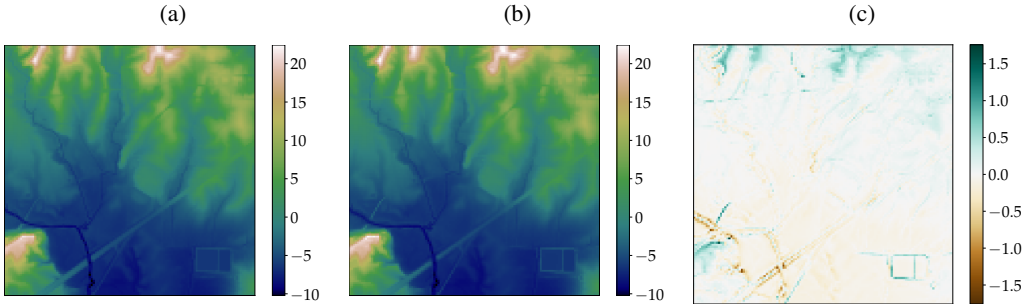

Figure A.10: Example where a narrow water passage (lower left) is expanded. (**a**) – Baseline coarse grid elevation map (**b**) – DNN coarse grid elevation map (**c**) – The correction of the DNN to the baseline. Colorbars in meters.

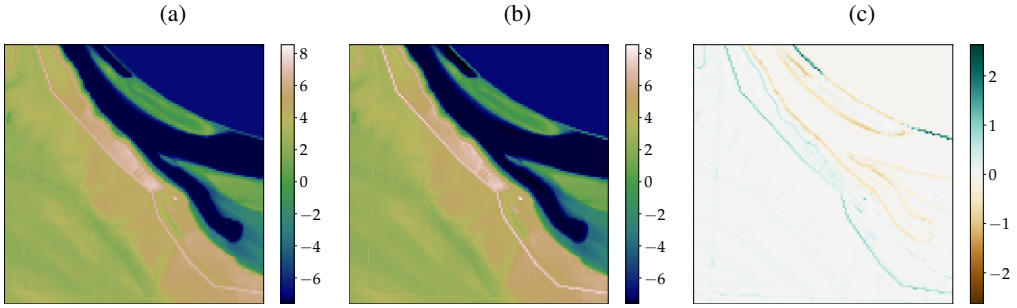

Figure A.11: Example where an embankment height is preserved. (**a**) – Baseline coarse grid elevation map (**b**) – DNN coarse grid elevation map (**c**) – The correction of the DNN to the baseline. Colorbars in meters.

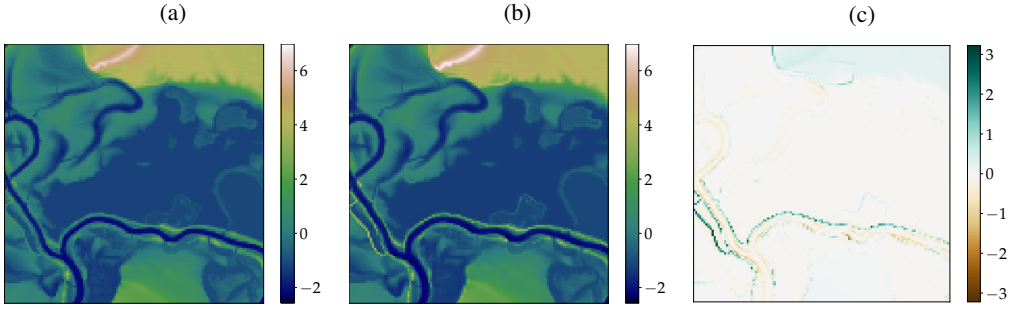

Figure A.12: Example where both river bank is expanded and embankments are preserved next to the river. (**a**) – Baseline coarse grid elevation map (**b**) – DNN coarse grid elevation map (**c**) – The correction of the DNN to the baseline. Colorbars in meters.