water

MDPI

Article

Bed Topography Inference from Velocity Field Using Deep Learning

Mehrdad Kiani-Oshtorjani * and Christophe Ancey

Environmental Hydraulics Laboratory, École Polytechnique Fédérale de Lausanne, 1015 Lausanne, Switzerland; christophe.ancey@epfl.ch

* Correspondence: mehrdad.kiani@epfl.ch; Tel.: +41-21-693-2378

Abstract: Measuring bathymetry has always been a major scientific and technological challenge. In this work, we used a deep learning technique for inferring bathymetry from the depth-averaged velocity field. The training of the neural network is based on 5742 laboratory data using a gravel-bed flume and reconstructed velocity fields, namely the topographies were obtained from real-world experiments, and the velocity fields were estimated using a statistical model. To examine the predictive power of the proposed neural network model for bathymetry inference, we applied the model to flume experiments, numerical simulations, and field data. The results showed the model properly estimates topography, leading to a model for riverine bathymetry estimation with a 31.3% maximum relative error for the case study (confluence of the Kaskaskia River with the Copper Slough in east-central Illinois state, USA).

Keywords: bathymetry estimation; deep learning; U-net architecture; entropy-based models; inverse modeling; gravel-bed flume/river

Citation: Kiani-Oshtorjani, M.; Ancey, C. Bed Topography Inference from Velocity Field Using Deep Learning. *Water* **2023**, *15*, 4055. https://doi.org/10.3390/w15234055

Academic Editor: Giuseppe Pezzinga

Received: 13 October 2023
Revised: 17 November 2023
Accepted: 20 November 2023
Published: 22 November 2023

1. Introduction

Imaging riverine bathymetry is fraught with difficulty. Among the available techniques in riverine bathymetry, a typical example is provided by the use of light detection and ranging (LiDAR) systems [1–3]. This technique is based on the estimation of distance by recording the time lapse between sending and receiving a signal and having prior knowledge of the signal's speed. While these techniques do provide direct access to bathymetry, it is time-consuming and costly. Other examples include the use of indirect bathymetry techniques [4], which usually involve solving an inverse problem relating bed topography to surface elevation [5,6], and surface velocity data [7,8].

Inferring variation in riverine bathymetry from surface flow data is predicated on a potent causal relationship between them [9]. This assumption is true, especially in shallow and one-dimensional flow conditions when the magnitude of velocity in the vertical direction is low compared to the streamwise velocity component [10]. Employing surface velocity data is an alternative to estimating bathymetry, thanks to its affordability and sensitivity to river depth [10–13]. There are several techniques for estimating surface velocity, including the use of drifter global positioning system (GPS) recordings [11,14,15]. For example, Landon et al. [11] investigated whether drifters' trajectories are sensitive enough to bottom topography to allow for depth determination. They successfully extracted river bathymetry using velocity field measurements collected from drifter GPS records in an ensemble-based data assimilation approach. Estimated bathymetry based on this technique on a shallow braided and deep meandering reach of the Kootenai River in Idaho in the United States of America (USA) was more accurate than the previous estimations. Wilson and Özkan-Haller [10] applied this technique to one-dimensional (1D) channels and for two real-world reaches: the Snohomish River, Washington, and the Kootenai River, Idaho, in the USA. The main difference with Landon et al. [11] was this: they used depth-averaged

velocities (based on numerical solutions to the shallow water equations) and a least-square method to minimize a cost function that combines known information and measured data. By using a state augmentation technique, the measured variable is connected to the unknown parameter, providing a model for deep-water bathymetry estimation (with a depth in the 3–10 m range).

In recent years, deep learning has become one of the most powerful tools for overcoming some deterministic approaches' limitations [13,16–18]. It can be used to image bed topography from surface flow data. Due to their ability to identify patterns or trends in data, neural networks are becoming increasingly popular in geophysics and hydraulics [19]. For instance, Ghorbanidehno et al. [13] used the neural network technique to obtain bed topography based on the depth-averaged flow velocity field, using limited labeled data. In order to reduce network size, the authors combined a fully connected deep neural network with principal component analysis (PCA). PCA is a widely recognized method for reducing the dimensionality of data, where it transforms the data onto a new basis with fewer dimensions [20]. For instance, this could be achieved by calculating the covariance matrix of the training data and extracting its eigenvalue decomposition.

Usually, training the network needs a huge amount of data to avoid the curse of dimensionality [21], which implies that the data occupy less and less of the data space as the data space moves from lower to higher dimensions. The volume of this space grows so fast that the data cannot keep up and thus become sparse—the sparsity problem is a major statistical significance issue. To enhance the training dataset's size, Ghorbanidehno et al. [13] divided the river's entire domain into small subdomains. As a result, each river profile provided several hundred training samples rather than just one.

The training of neural networks for bathymetry estimation can be performed by solving the shallow water equations and using the resulting solutions. A typical example is provided by Liu et al. [17], who employed shared encoders and separate decoders, where bathymetry's input image is encoded and then decoded to three outputs, namely the flow's longitudinal and transverse depth-averaged velocity components and the water surface elevation. Two-dimensional (2D) simulations using randomly generated input bathymetry data were used to generate the training data.

Recent advancements in measurement techniques have led to the availability of high-resolution and low-cost surface velocity fields [8]. These techniques include the use of Lagrangian drifters [14,22], large-scale particle image velocimetry [23,24], and synthetic aperture radar [25,26]. This work intends to predict bathymetry by using indirect measurements, i.e., the velocity field through a convolutional neural network. The neural network training is based on a U-net architecture, used for the first time for segmentation problems [27], and, to our knowledge, this is the first attempt to use the U-net model for bathymetry inversion. To this end, we used an experimental dataset made of 5742 data. It was not realistic to measure the velocity field, but we could estimate it using entropy-based models. Figure 1 shows the model's architecture. In this network, the encoder and decoder are connected by skip connections—in contrast with regular convolutional networks. By connecting them in this way, we ensure we actually lose no information during the feature extraction process [28,29]. In order to develop a well-trained neural network model, we need to collect a large amount of data. Due to the size of the experimental dataset used in this study, we had no problems with dataset size. As an alternative, if we were dealing with a small dataset, we could divide the experiment domain into smaller regions and thereby increase the amount of data.

In this work, training the convolutional neural network model to infer bathymetry is based on bed topography scans and velocity field estimates. Solving the governing partial differential equations will be bypassed and the solutions inferred solely by a convolutional neural network that is trained based on a huge experimental dataset. Closely related to our work is Ghorbanidehno et al. [13]'s PCA-DNN framework, which combined the traditional fully connected deep learning method and principal component analysis. They trained the deep neural network model based on field data, whereas we focused on a convolutional

neural network; moreover, our dataset is produced in the laboratory. Our trained model is suitable for gravel-bed rivers whereas their model is suitable for deep rivers. All training of the networks in this work has been performed using Colaboratory on GPU: Nvidia Tesla P100-PCIE-16GB, manufactured by Nvidia Corporation, with headquarters in Santa Clara, California, United States.

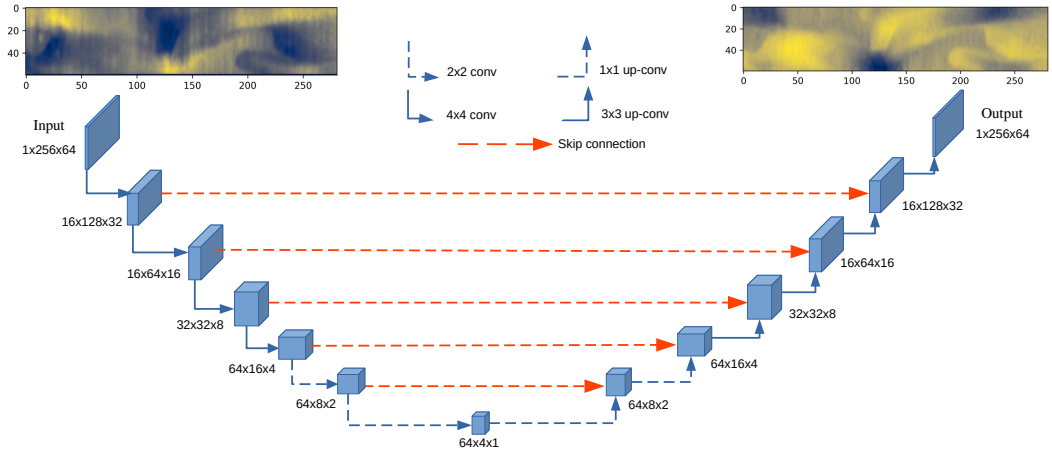


Figure 1. A diagram showing the neural network architecture. Convolutional layers are shown in blue, and skip connections are shown in orange. The numbers in the format of $a \times b \times c$ close to each box indicate the number of channels and the image size in the x and z directions, respectively.

This paper is structured as follows: The methodology for generating the dataset based on experiments and analytical derivation is explained in Section 2. Moreover, the U-net architecture is described and the basic parameters, regularization, and hyperparameters—such as dropout, learning rate, weight decay, and training dataset size—are studied in that section. In Section 3, the model's performance and accuracy have been studied. In addition, the neural network model's predictive power has been studied by applying it to flume experiments, the numerical simulation of a gravel-bed flume, and field data performed at the confluence of the Kaskaskia River and Copper Slough in east-central Illinois state, USA. Sections 4 and 5 discuss and summarize our achievements and future work, respectively.

2. Methodology

2.1. Data Generation

The dataset is based on the experiments conducted at LHE-EPFL [30]. Table 1 represents the input parameters and each experiment's duration. It consists of three long-term experiments. Experiments were carried out in a tilted flume at an angle of $S = 1.6\%$ and 1.7%, a length of $L = 17$ m, and a width of $w = 60$ cm, as depicted in Figure 2. The useful length of the flume is 14 m because of technical limitations (1.5 m from each side is ignored). The flume bed was made of natural gravel with a height of 31.5 cm at the beginning of the experiments. Sediments' mean diameter was $d = 5.5$ mm with a $\sigma = 1.2$ mm standard deviation and a density of $\rho_s = 2660$ kg/m^3. The water discharge and sediment feeding rates at the flume inlet were set to 15 L/s and 2.5, 5, and 7 g/s, respectively. Each experiment lasted for hundreds of hours. At the beginning of each experiment, the bed surface was flattened. These long-term experiments are conducted via a sequence of short-term experiments that last 8 hours. To prevent the destruction of the bed topography, these short-term experiments are launched with very low flow discharge. During the experiment, the water discharge and sediment feeding rates were kept constant [30]. Experiment 1 included 1566 scans, and experiments 2 and 3 consisted of 741 and 3435 scans, respectively (in total 5742 scans, each consisting of bed topography and flow depth data). Based on the entropy-based models, we computed the depth-averaged velocity field using the flow depth data and knowing the constant flow discharge as a constraint. In the next following sections, the details of entropy-based models for inferring the velocity fields will be explained.

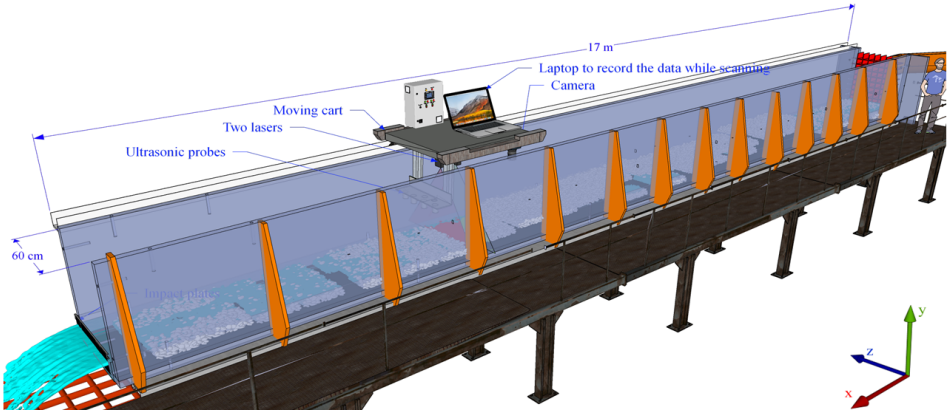


Figure 2. Gravel-bed flume illustration in which all of the dataset has been gathered based on the experiments performed on this channel. It has a 17 m length, but the useful length (removing 1.5 m from each side) is 14 m.

Table 1. Input parameters of the experiments.

	Exp. 1	**Exp. 2**	**Exp. 3**
Flow rate (L/s)	15	15	15
Flume slope (%)	1.6	1.7	1.6
Sediment feed rate (g/s)	2.5	7.5	5.0
Duration (hours)	250	556	118

During an experiment, measurements of the bed topography and flow depth are recorded with a fine resolution. The bed topography is measured by laser-sheet imaging technique, i.e., using two angled lasers. Those lasers are mounted on an automated moving cart. Water disturbances, caused by a variety of factors (including turbulent flow conditions, varying bed topography, sediment transport, and inflow/outflow perturbation) can interfere with measurements. To overcome this issue, calibration is conducted prior to the experiments. By calibrating the distance between two angled lasers projected on the bed, the bed topography can be induced with a resolution of 60 $\times$ 281 pixels. Every 10 min, the cart programmed to scan the bed proceeds using MatLab. Each scan takes about 145 s and covers 14 m $\times$ 60 cm. During each scan, the bed topography is measured by lasers, and the flow depth is measured by ultrasonic probes. The ultrasonic probes determine flow height; a sound pulse is emitted and the traveling time between the sensor and the object can be used to infer flow depth (see Figure 2).

The sediment feeding system operates as follows: It moves sediments from the hopper to the flume's inlet via a conveyor belt. Subsequently, these sediments are introduced into the channel through a pin board, which spreads the gravel along the width with a Gaussian distribution. The rate at which sediment is fed is regulated by controlling the speed of a rotating cylinder that clogs up the hopper's outlet. The most frequent mode happening in the flume using the aforementioned flow, channel geometry, and sediment characteristics is alternate bars. In Figure 3, the bed topography and alternate bars' appearance are illustrated.

Due to erosion, deposition, and transport processes, the bed topography changes over time during the experiments. In turn, the hydraulic conditions in the system are affected by the bed morphology, as a result of a feedback loop driven by the movement of particles [30,31]. Based on the Froude number, $Fr = v/\sqrt{gh}$ (where g is the gravitational acceleration, v is flow velocity obtained by particle image velocimetry using tracking light polystyrene balls, and h is flow depth measured by ultrasonic probes), the flow regime was turbulent and super-critical with a Froude number of more than 1 [30]. Since the flow depth is uniform at the beginning of each experiment—when there is no bed form— the streamwise bed shear stress can be measured by $\tau = \rho g R_h S$ where $R_h = hw/(2h + w)$ is

the hydraulic radius, and ρ is fluid density. The shear stress is estimated by computing its average value along longitudinal bed profiles [30,32]. The value of shear stress in our experiments varies from 4.89 to 10.75 kg/m/s^2. The Shields number and the shear velocity can be obtained by $\tau^* = R_h S/R/d$, and $u^* = \sqrt{\tau/\rho} = \sqrt{gR_h S}$, respectively, where $R = (\rho_s - \rho)/\rho$. The shear velocity varies in a range of $0.07 \leq u^*(\mathrm{m/s}) \leq 0.10$, and the Shields number changes according to $0.05 \leq \tau^* \leq 0.11$ among all experimental data.

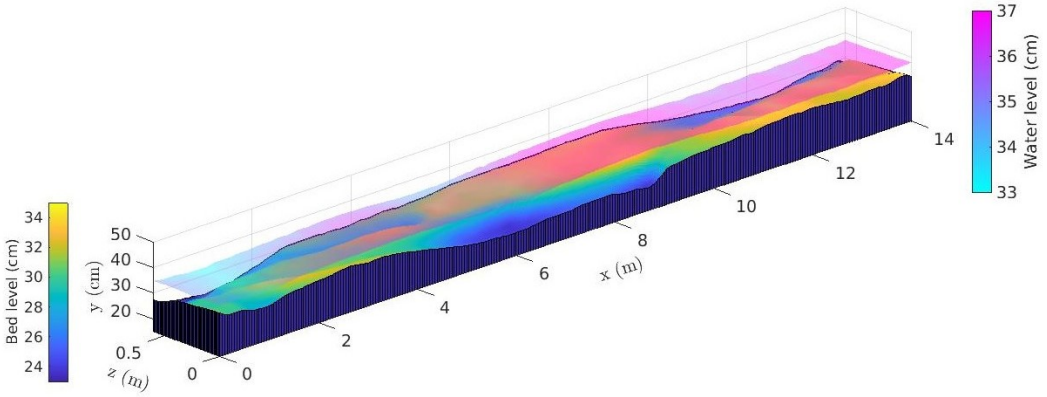


Figure 3. An example of experimental topography using the gravel-bed flume. The bed and flow heights are measured from the bottom of the flume. The initial height of the bed is 31.5 cm.

In order to have a suitable input/output image size for training the neural network, we applied a bicubic interpolation over a 4 × 4-pixel neighborhood to the flow depth and bathymetry data to change the size of the data, i.e, the bed topography and velocity fields from 60 × 281 pixels to 64 × 256 pixels. Subsequently, the dataset was partitioned into three subsets: the training dataset, the validation dataset, and the test dataset. The distribution of the total number of samples was as follows: 80% for training data and 20% for validation data. Moreover, the training data were again divided into 90% as the training and 10% as the test dataset. Training and validation data will be used to fit the models and to estimate the prediction error for model selection and hyperparameter tuning, respectively, and, finally, a test dataset will be used to assess the generalization error of the selected (best) model. The dataset is composed of 5742 pairwise (bed topography and velocity field) grayscale images, and, therefore, the training, validation, and test datasets consist of 4133, 1149, and 460 data, respectively. The dataset is published attached to this work on https://rb.gy/84l4hk, accessed on 19 November 2023. A total of 11,484 fields are included in the published dataset, including both bed topography and depth-averaged velocity data.

2.2. Entropy-Based Velocity Profile

Entropy is defined as a measure of a system's randomness or disorder and originated from thermodynamics [33]. Shannon was one of the pioneers who developed entropy's mathematical foundation and connected it to information [33]. To obtain the one-dimensional and two-dimensional velocity profile, it is necessary to maximize the entropy of the velocity distribution in order to obtain the least biased velocity probability density function [34]. It is based on Jaynes's principle of maximum entropy, which states that any system in the equilibrium state attempts to maximize its entropy, subject to given constraints [35]. The entropy of a flume/river must reach its maximum value when it reaches a dynamic (or quasi-dynamic) equilibrium [36]. At any location where maximum velocity occurs, the 2D velocity distribution based on entropy theory should be valid [33].

As aforementioned, during each experiment, bed topography and flow depth are measured locally with high resolution. Since the flow discharge is fixed to a constant 15 L/s value during all the experiments, the constraint in the entropy-based models is constant flow discharge. Then, the velocity profile can be obtained locally by maximizing entropy. Estimating flow velocity can be achieved using a number of techniques. Two

entropic principles are known as Shannon and Tsallis (this entropy is a generalization of the Shannon entropy), which are often applied to river discharge assessment [33,37]. Both of these principles connect the maximum flow velocity at a vertical axis of the flow area to the cross-sectional mean flow velocity.

2.2.1. Shannon Entropy-Based Method

The usage of Shannon entropy to the open-channel flows has been shown by Chiu and colleagues [38–40]. Entropy-based models have been in use in hydraulics for a number of years, and we have some hindsight about their validity [34,41]. Building upon Chiu's work, Moramarco et al. [42] proposed a version of Chiu's entropy-based velocity distribution equation that can calculate the 2D velocity distribution using only the maximum velocity and bathymetry information [34]. This model does not require parameter calibration or the isovel equation and only requires knowledge of the maximum velocity and its position. Generally, the maximum flow velocity occurs at the water's surface; especially when $w/h > 3.5$, the maximum velocity occurs on the flow surface [43], which is true of our experiments. The velocity equation can be expressed as follows [34,42]

$$u = \frac{u_{max}}{M} \log\left[1 + \left(e^M - 1\right)\frac{y}{h - D} \exp\left(1 - \frac{y}{h - D}\right)\right] \quad (1)$$

The equation for estimating the longitudinal velocity along the vertical axis (u) includes several variables, namely the maximum flow velocity, and its depth from the flow surface are represented by u_{max} and D, respectively. The Shannon entropy parameter is symbolized as M, whereas the variable y signifies the vertical position of the velocity with respect to the bed topography. To determine the constant value of the entropy parameter, the following equation can be used

$$\frac{u_m}{u_{max}} = \frac{e^M}{e^M - 1} - \frac{1}{M} \quad (2)$$

The mean velocity over a cross-section u_m can be calculated if we know the flow discharge and the section area, while u_{max} is determined iteratively by knowing the maximum velocity happening at the water's surface. Water depth and bed topography during the experiments were measured at a high spatial resolution, as detailed in Section 2.1. Our measurement system provided matrices with dimensions of 60×281 elements (width by length). For each of these 16,860 elements, we computed the velocity field using entropy-based models and a depth increment of $dy = 0.1$ mm by employing the algorithm described below and using a trial-and-error approach. As a consequence, the size of the velocity vector at each of these 16,860 elements depended on the water depth at that point. The algorithm is as follows:

- Assume a maximum velocity (e.g., $u_{\max} = 1$ m/s) in the initial iteration;
- Calculate the section flow area A using the measured local flow depth data with the assistance of the trapz function in MatLab;
- Determine the mean cross-sectional flow velocity u_m by dividing the flow rate $Q_l = 15$ L/s by the section area A;
- Compute the entropy parameter M based on Equation (2);
- Iterate over each set of 60 data points along the channel width, and calculate the velocity profiles for the cross-section using Equation (1);
- Estimate the flow discharge $Q_{l,est}$ based on the calculated velocity profiles;
- Adjust the maximum velocity in the initial step and repeat the process until the error $err = Q_l - Q_{l,est}$ is less than 10^{-3}.

The contour plot of the calculated velocity profile is plotted in Figure 4a. However, for the sake of comparison, Tsallis's entropy-based method has been applied to the same cross-section (the methodology is explained in Appendix B). The velocity profile based on the Tsallis entropy is plotted in Figure 4b. One can see that the velocity gets close to 1 m/s on the surface flow, which coincides with the measured flow surface velocity in the

laboratory using the particle image velocimetry (PIV) technique by tracking polystyrene balls traveling along the entire flume length [30]. By comparing the two velocity fields in Figure 4a,b, there is no such difference in using each entropy model. We therefore decided to use the Shannon entropy to calculate the velocity fields in all of our experiments.

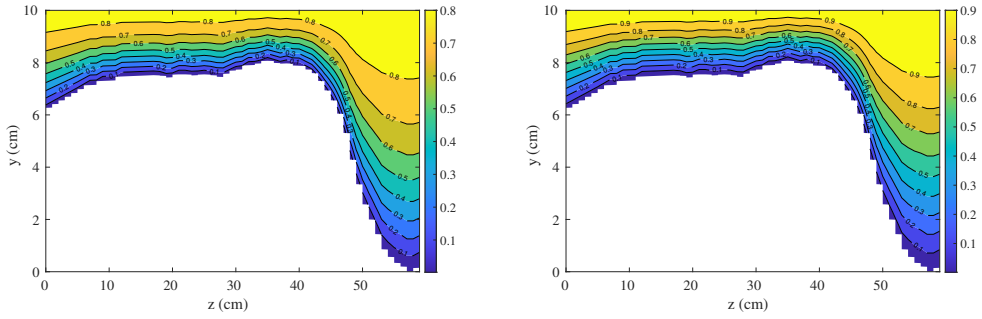


Figure 4. Velocity plot based on (**a**) Shannon entropy, and (**b**) Tsallis entropy for a cross-section in the middle of the channel based on exp. 3 at $t = 200$ min. The border between the white and blue regions indicates the bed topography for one cross-section.

2.3. Neural Network

We use the U-net architecture for our neural network model. In the main journal paper, in which the U-net was introduced by Ronneberger et al. [27], the input and output images were of different sizes and used to solve the segmentation problem. Since our input/output is of the same size, we reworked the model by changing the convolution function parameters and using the model as a regression problem. The U-net has been used for segmentation [44–46], classification [47–49], and the regression problem [50–52].

It has already been mentioned that the U-net is divided into two parts. Firstly, a standard convolutional neural network architecture is used to perform the contracting process. Leaky ReLU activation units are followed by multiple convolutions with padding in the contracting path. The same structure is repeated several times. One of U-net's key characteristics lies in the expansive path, i.e., the second path. Using transposed convolution, the feature map upsampled in each stage of the expansive path [27,53]. Afterward, we concatenate the upsampled feature map with the corresponding layer from the contracting path. Therefore, we obtain a U-shaped network, and, perhaps most importantly, contextual information is propagated along the network, enabling a proper reconstruction of context [27,53]. In this work, an upsampling followed by a convolution rather than a transpose typical convolution is used instead.

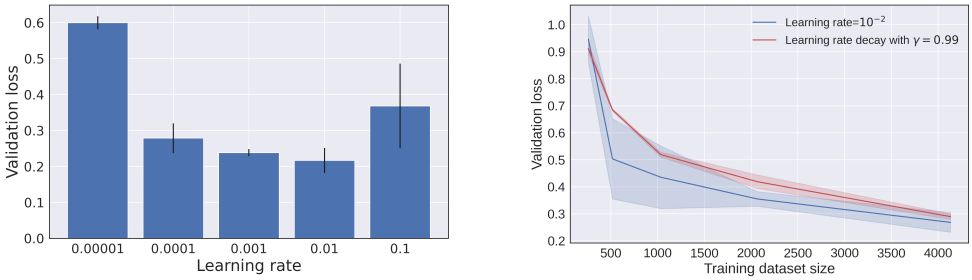
These features allow the model to handle the spatial complexity required for the pixel-wise regression tasks [54]. Its symmetric structure, combining both downsampling (reducing the spatial resolution of the feature maps while increasing their depth and detecting the important characteristics in the input image) and upsampling paths (identifying the features while maintaining the spatial resolution of the input), allows for the effective capture of multi-scale spatial information, which is essential for accurate regression predictions [27,55].

The implementation is based on the Pytorch deep learning framework [56]. Table A1 in Appendix A shows the structure of the U-net used in all our tests with details of the different layers. The schematic of our neural network architecture can be seen in Figure 1, in which a fully convolutional U-net is used. This is a famous architecture that uses a number of convolutions at various spatial resolutions. This network differs mainly from a regular encoder–decoder network in that skip connections are used from the encoder to the decoder parts. By using skip connections, the network can use fine-grained details learned in the encoder to reconstruct an image in the decoder [27,57]. No pooling is used, and, instead, strides and transposed convolutions are used.

The selection of U-Net's hyperparameters, such as learning rate, number of epochs, batch size, number of layers, activation functions, optimizer, dropout rate, and weight initialization, has a substantial influence on its efficacy. Traditional procedures for hyperparameter optimization, such as grid search or random search [58], can be used, but they are computationally costly. To solve this issue, some recent research has proposed methods for refining the U-Net hyperparameters [59]. In this study, we used a variety of tests to establish the best hyperparameters explained in Sections 2.3.1 and 2.3.2. However, for some hyperparameters, such as batch size, the selection method involves trying out several values and selecting the best one by using a random search strategy.

2.3.1. Basic Parameters

In order to tune the training hyperparameters such as learning rate and learning rate decay, we have studied these hyperparameters, following Thuerey et al. [50], using the baseline architecture. The learning rate is a tuning parameter of the optimizer function that determines the step size taken by the optimizer at each iteration while moving toward a loss function minimum. Figure 5a shows the validation loss using different learning rates varying in a range of $[10^{-5}, 10^{-1}]$. The validation loss is computed by employing the mean absolute error (*MAE*-*L*1 loss) as the metric, which measures the disparity between the bathymetry field $\hat{y}$ derived from the validation dataset and the bathymetry field predicted by the neural network model, denoted as y. The *MAE* is calculated by summing the absolute differences between the predicted values (y) and the actual values ($\hat{y}$), divided by the total number of validation dataset size, denoted as N being expressed as: $MAE = 1/N \sum_{i=1}^{N} |\hat{y}_i - y_i|$. The error bar, which represents result uncertainty, is calculated by repeating the same configuration four times. As can be seen, it is clear that the lowest and largest learning rates result in the highest validation loss. Considering our problem, we have the best learning rate at 10^{-2}.



(**a**) (**b**)

Figure 5. (**a**) Effect of different constant learning rates in terms of validation loss, (**b**) testing models based on validation loss with and without learning rate decay versus training dataset size.

With gradient-based optimization algorithms, saddle points can be approached, and training takes a longer time since the surfaces around such points are flatter and the gradients are close to zero [60]. Therefore, rather than using a fixed value for the learning rate, it could be decreased over time. If training no longer improves our loss, the learning rate is changed; every iteration is based on some cyclic function f. The number of iterations in each cycle is fixed. By using this method, the learning rate is allowed to vary cyclically between reasonable boundaries. We can traverse saddle point plateaus more quickly when we increase our learning rate since we are less likely to become trapped in unfavorable states such as saddle points. We have performed a test: in it, the learning rate decays by a gamma factor in every epoch that is set to be $\gamma = 0.99$, i.e., $lr_{epoch} = \gamma \times lr_{epoch-1}$. Figure 5b compares validation loss based on this strategy—comparing constant and varying learning rate factors. We found out that the learning rate decay does not help the improvement of the model based on validation loss comparison; for nearly all training dataset sizes, the validation loss when using learning rate decay is greater than that with a constant learning rate, and it helps decrease diversity around the mean value.

Based on the tests in Figure 5a,b, we used the Adam optimizer [61] with a fixed learning rate of 0.01 and $\beta_{1,2} = [0.5, 0.999]$ for the rest of the training optimizations. In addition to a nonlinear activation function, each of the network sections contains a convolutional layer, batch normalization, and dropout layer (refer to Table A1 in Appendix A). In the model, we included the possibility of normalizing batches. During training, it shifts and rescales according to the mean and variance estimated on the batch. The literature has proven that batch normalization makes the training process faster and smoother [62,63]. However, it requires a sufficiently large batch size, and our choice of batch size is 100. We did not use any pooling for the training process.

2.3.2. Regularization and Hyperparameters

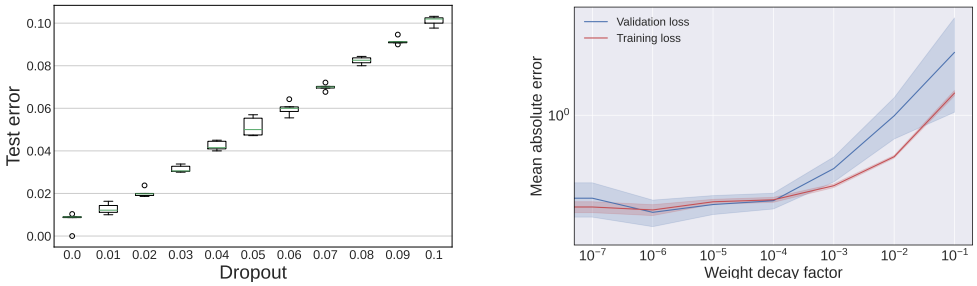
The idea behind regularization is to reduce the complexity of the neural network model and inhibit overfitting—that is, the model learns unnecessary details, noises, or random fluctuations in the training data as main concepts, insofar as it negatively impacts the model's performance. It is essential to study this when using deep learning algorithms. For example, dropout, weight decay, data augmentation, and weight initialization are the most often used regularization techniques. Following Thuerey et al. [50], these techniques are studied and discussed in the subsequent analysis.

The choice of initialization is highly influential on deep learning algorithms when training deep models. Algorithm converge-ability can be determined by the initial point, with some initial points being so unstable that numerical difficulties arise and the algorithm fails. The weight initialization of a neural network consists of setting its weights at small random values, which determine the starting point of the optimization when training. In order to prevent layer activation outputs from exploding or vanishing gradients during training, an appropriate initialization of the weights is necessary and manages to achieve better performance. The following are the most widely used, among others, when dealing with convolutional layers: the constant initialization, the He normalization [64], and the Xavier normalization [65]. In deep networks, the performance of the constant initialization scheme usually is subpar [60]. We chose not to use constant initialization because it sets all computing units to the same state, resulting in symmetric or identical outputs and gradients during backward propagation. This symmetry restricts the flexibility of our network; hence, we chose not to use constant initialization. We experimented with two other initializations for our networks: the He and Xavier normalizations. The former seems to lead to slightly better performances than the latter. We therefore stuck to that one. Based on the He initialization method, initialization is based on a randomly generated number computed with a Gaussian probability distribution with a mean of zero and a standard deviation of $\sqrt{2/n}$, where n stands for the number of inputs to the node [64].

We can go one step further by adding mechanisms specifically designed to facilitate the training, such as dropout [66]. Dropout consists of randomly dropping out neurons in a layer, with a probability that corresponds to the dropout coefficient. We implemented the possibility to include it in the model. This technique's main advantage is that it prevents overfitting as the neurons of a layer become less dependent on particular inputs. In Figure 6a, the influence of the dropout is studied, which has a negative effect on the test error. As the dropout rate decreases, the test error tends to converge to zero. Therefore, the dropout is not recommended for this study, and it is considered to be zero for the rest of the training. The reason may be that dropout is usually unnecessary when the network is small compared to the dataset. Adding this regularization will worsen the performance if the model's capacity is already low.

In order to further improve the models' performance, we can also use the weight decay technique, which holds greater significance for fully connected neural networks compared to convolutional neural networks. This approach consists of adding a penalty to the model based on the amplitude of its weights in order to limit overfitting [67]. The model will be penalized all the more as the values of the network connections increase. This strategy is based on the principle that large weights in a neural network can cause more variance at

the output and prevent the model from generalizing correctly. The penalty forces weights down and permits a less flexible network that is less specialized in the data used for training. This penalty is defined as follows: $L_{weight\_decay} = \lambda \sum_i \omega_i^2$, where L_{weight_decay} is the penalty associated with weight decay, ω_i is the i$-th$ weight in the network, and λ is a positive coefficient that affects the importance of the penalty [68]. This parameter has yet to be determined, so we carry out training with several values of λ as presented in Figure 6b, in which the validation and training loss are plotted as a function of the weight decay factor. Losses for a weight decay factor larger than 10^{-4} start to increase while losses for a lower weight decay reach a plateau. The training time using a different weight decay factor does not change dramatically; therefore, the best weight decay can be chosen by just paying attention to the losses in which 10^{-6} is the best for the rest of the training.



(**a**) (**b**)

Figure 6. (**a**) Test error as a function of the dropout rate for a model with 142,689 training parameters, (**b**) illustrates the influence of the weight decay factor on the training and validation performance.

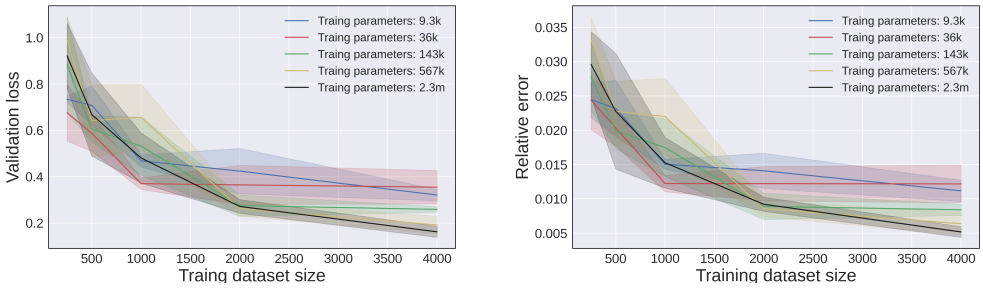
Data augmentation is a technique in deep learning methods that increases the amount of a training dataset by leveraging existing data, often via techniques involving alterations in position and color, among others. In physical applications, it is also feasible to employ data augmentation, as presented in Ghorbanidehno et al. [13]. Nevertheless, as evidenced by the results detailed in the subsequent subsection, when evaluating the adequacy of data volume relative to the model's complexity through validation error comparison, it becomes apparent that the existing data volume suffices. Consequently, in this study, we refrained from employing any data augmentation techniques.

3. Results and Performance

3.1. Model Performance

To investigate the impact of a neural network model's complexity on the accuracy of its predictions, in the following training runs, the total amount of weights is modified by varying the number of feature maps. For instance, the total weights will quadruple by duplicating the number of input channels [50]; hence, by manipulating the number of input/output channels, we can investigate diverse model complexities. Figure 7 shows the accuracy results for five distinct network sizes, namely networks with $\{9.3, 36, 143, 567, 2300\}$k training parameters, as a function of various dataset sizes including $\{0.25, 0.5, 1, 2, 4\}$k data.

A network with 9.3k training parameters is relatively small for a generative neural network with $1 \times 64 \times 256 = 16$k outputs, but it allows for a faster training time and prevents overfitting in view of the relatively small dataset with which we are working. Nevertheless, a network comprising 2.3 m training parameters faces challenges concerning time constraints and lacks generalization capabilities due to its relatively small training dataset. The number of training parameters is a crucial number to keep in mind when training neural networks. It is easy to change and increase it and end up with a network with millions of parameters; then it is highly probable that we will be faced with all kinds of convergence and overfitting issues. To avoid the dimensionality curse, the number of parameters must match the amount of training data, as well as scale with the network's depth [69]. The exact relationship between these three depends on the problem under consideration.



(**a**) (**b**)

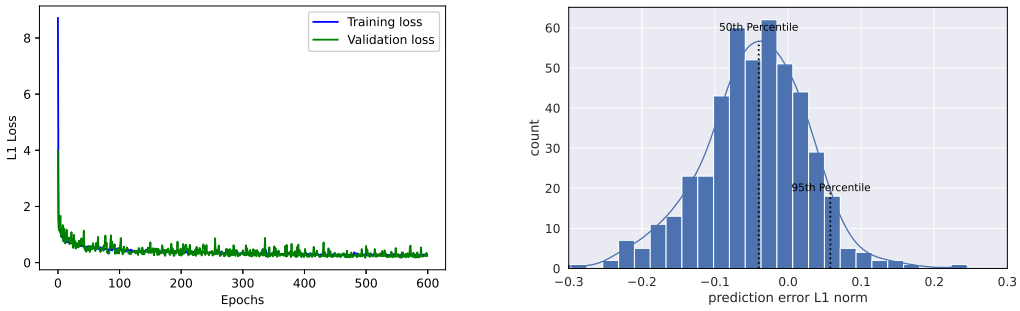
Figure 7. (**a**) Validation loss and (**b**) relative error for various model complexity and training dataset size.

In Figure 7a, various models with different complexities—the number of training parameters—perform differently when applied to 1149 velocity fields from the test dataset. As depicted in Figure 7a, both the error and error bar tend to be larger when training occurs on a small dataset size. Conversely, as the dataset size increases, the error and error bar decrease. Furthermore, for a large dataset size, increasing the number of training parameters leads to a decrease in the validation error. However, for a small dataset size, augmenting the number of training parameters results in an increase in the validation error due to overfitting to the data.

In Figure 7b, the relative error (RE) is computed using the formula: $RE = 1/N \sum_{i=1}^{N} |y_i - \hat{y}_i|/\hat{y}_i$, where $\hat{y}$ refers to the ground-truth bed elevation, y represents the estimated bed elevation calculated based on the neural network model, and i counts the sample data. The curves in Figure 7b show the relative errors for various model sizes and training dataset sizes. For a given number of training samples, there exists a slight variation among the relative errors of different model sizes. However, the decrease in error is comparable when larger amounts of training data are utilized. Once again, the error bar reduces as the dataset size increases. When dealing with the smallest dataset size, an increase in training parameters leads to a corresponding increase in relative error. However, this behavior reverses for larger dataset sizes. Based on the findings in Figure 7, the most optimal model seems to be the one with 2.3 m training parameters, achieving an $L1$ loss of 0.16 cm and a relative error of 0.5% when trained on a 4k dataset size. However, this model requires a considerable training time of approximately 2 h. In comparison, smaller models show faster training times, taking $\{0.31, 0.39, 0.59, 0.97, and 1.8\}$ hours—from the smallest to the largest model for 600 epochs-, when applied to a 4k dataset size.

Considering both the execution time for training and the performance metrics, we have ultimately chosen the model with 143k trainable parameters as the best model. This model exhibits a relative error of 0.85% and a validation loss of 0.26 cm when trained on the 4k dataset size. These results demonstrate that the model has been effectively trained and can generate accurate predictions within a reasonable timeframe.

Figure 8a shows the best neural network is trained properly without overfitting—given that the training and validation loss plots did not exhibit divergence. After 600 epochs, the validation and training losses should have decreased from an initial value of around 8.9 cm to 0.02 cm based on the standard settings. It is visually easy to see the loss curves trend downward for 100 epochs, and, afterward, the curve flattens out. As we move toward the end by increasing the number of epochs, the validation loss is still decreasing slowly, and, most importantly, it is not increasing. Nevertheless, achieving a training loss reduction of more than 0.02 cm will come at the expense of increased training time without a significant improvement in the loss compared to previous epochs. A divergent validation loss from the training loss would indicate overfitting, which is something we should avoid. The graph illustrates that the models do not exhibit overfitting over time and reach stable levels of validation and training losses after 600 epochs.



(**a**) (**b**)

Figure 8. (**a**) Training and validation losses for learning rate 0.01; (**b**) histogram plot of the $L1$ norms of prediction errors (applied to 460 test samples).

To study the neural network model performances, in which we have tuned the parameters in the previous parts on a larger number of training and validation datasets, we perform an error measurement for all the test datasets. The best model is applied to the test dataset, and the $L1$ error is calculated. Figure 8b shows the distribution of the $L1$ error varying in a range of $[-0.32, 0.24]$ cm. The 50th (or median) and the 95th (meaning that only 5% of the samples exhibit a greater $L1$ error) percentiles are -0.041 cm and 0.058 cm, respectively, with a mean of -0.045 cm. This means that the $L1$ error of the examples in the test dataset predicted by the neural network model lies close to zero. The maximum error is 3.2 cm, and the bed topography changes at an interval from 18.19 cm to 37.02 cm, meaning in an 18.83 cm range.

3.2. Model's Predictions

In this section, the best-trained network's prediction capability may be investigated further. By selecting the best model, namely the one with 143k training parameters, no dropout, and a learning rate of 10^{-2}, and by choosing 10^{-6} as the weight decay factor and using the whole dataset, i.e., 4133 data points for the training, the final neural network is established and used for studying accuracy and performance in different case scenarios, i.e., the test dataset, a numerical simulation, and real-world case scenarios.

3.2.1. Model's Predictions Based on Experiments

A part of the dataset (460 data points) is kept for testing the neural network model's prediction power. The velocity fields in the test dataset feed into the neural network and the prediction is examined by comparing them with the ground truth (experiments). The accuracy of the model is shown in Figure 9 so that each column represents one of the bathymetry in the test dataset. The first row is the ground truth of the bed topography, and the second row shows the predicted bathymetry using the neural network model. The differences between the aforementioned topography fields (ground truth—prediction) are presented in the third row. The mean error among all of the test datasets is 0.045 cm, while the topography varies in a range of 18.8 cm.

To quantify the model's prediction further, the topography over a longitudinal axis along the flow direction is compared in Figure 10 with the neural network model's predictions. The data presented from the first to third columns of Figure 10 correspond to the respective data from the first to third columns of Figure 9. The longitudinal axes are located $z = 5$, 30, and 55 cm from the down wall. The black and red solid lines represent the prediction and the ground truth (experimental data), respectively. As can be seen, the curves' variations are finely reproduced, and the values coincide very well. This allows us to conclude that hyperparameters related to the training and the architecture of the neural network model are chosen properly and provide sufficient accuracy. In addition to the validation of the model based on the test dataset, the numerical simulation performed

and field data measurements are used for comparison with the model's prediction in Sections 3.2.2 and 3.2.3, respectively.

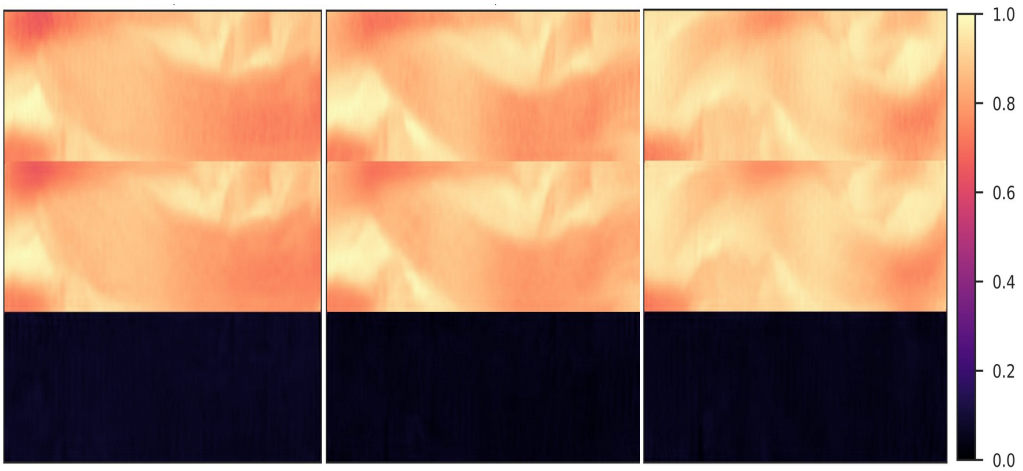


Figure 9. Qualitative result of the U-net: the first row is the normalized bed topography fields from the test dataset (ground truth). The second row contains the images that are predicted by the U-net. The third row contains the error images (ground truth—prediction).

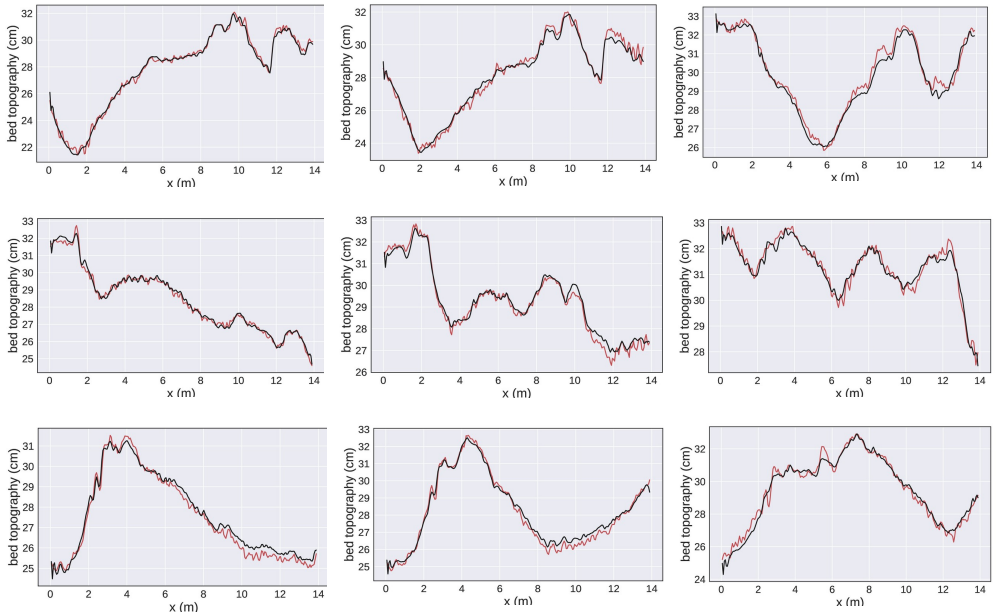


Figure 10. Each column belongs to one experiment. The first row is the profile of the bed topography along the channel length at $z = 5$ cm from the down wall, the second row is the profile at $z = 30$ cm, and the last row is the bed topography at $z = 55$ cm. Ground truth is shown by red, and prediction by black color.

3.2.2. Model's Predictions Based on Numerical Simulation

In Section 3.2, the model was applied to the test dataset derived from experiments conducted at LHE-EPFL [30], and it yielded precise predictions. Since the model is trained on the same experimental dataset (the training part of the dataset), the model prediction using the test dataset in Section 3.2.1 just shows us the model is well trained and that, if someone applied the model to the gravel-bed laboratory channel, they could obtain very accurate results. An important question would be about the model's accuracy in the other applications. In order to further study the model's prediction power, the model will be compared with the simulation performed based on Iber 2.5.1–hydraulic software for the simulation of free surface flows [70].

A gravel-bed flume is simulated based on Iber. A flume with a $Q = 15$ L/s flow discharge, 1% slope, 10 m length, and 0.5 m width, using a uniform structured mesh grid with 100 elements in the flow direction and with 10 elements in the transversal direction, has been simulated with a final time of $t_f = 7200$ min. The virtual flume dimensions differ from the dimensions of the flume on which the model is trained (0.6 m by 14.0 m). In spite of the difference in size, the trained model can be applied to the simulation flume. Additionally, using the resize function from the OpenCV library with bicubic interpolation, the image size is increased to 256×64 elements, allowing it to be fed into the ML models. The mean particle diameter is considered 7 mm in addition to using a Manning coefficient of 0.025 and choosing Meyer-Peter and Müller (MP-M) as the bedload model on the software. The simulation input parameters are intentionally selected to be close to the input parameters of the experiments upon which the neural network is trained.

The inlet boundary condition is set to a constant flow discharge, and an open boundary condition for the output has been chosen. Afterward, the velocity field and the topography at the end of simulation time, namely $t_f = 7200$ min, are extracted from Iber. The trained neural network model has been applied to the depth-averaged velocity field extracted from Iber, and the topography prediction from the neural network model has been compared with the topography of Iber.

The top plot in Figure 11 is the channel bathymetry based on numerical simulation, and the down plots are a comparison of the neural network prediction versus the bed profile based on Iber on a cross-section at $x = 1.0$ m (left plot) and $x = 5.5$ m (right plot) from the flume inlet. Figure 11 shows the neural network model's prediction in comparison with the ground truth (bathymetry from Iber), in which the prediction has a 25% maximum relative error. The mean absolute error in the cross-section $x = 1.0$ m is 0.75 cm, while, at the cross-section $x = 5.5$ m, it is 0.14 cm. However, gaining such a mean absolute error is small in comparison to the range in which the bathymetry changes. This validation shows that, as long as the flow and bed satisfy the mass and momentum conservation laws, the trained model is robust and accurate predicts bathymetry for gravel-bed and shallow flows. Despite finding out about the neural network model's performance in laboratory experiments and simulation, inquiring about model performance on the field data is a natural question and an intriguing subject to examine. It will be answered in the next part.

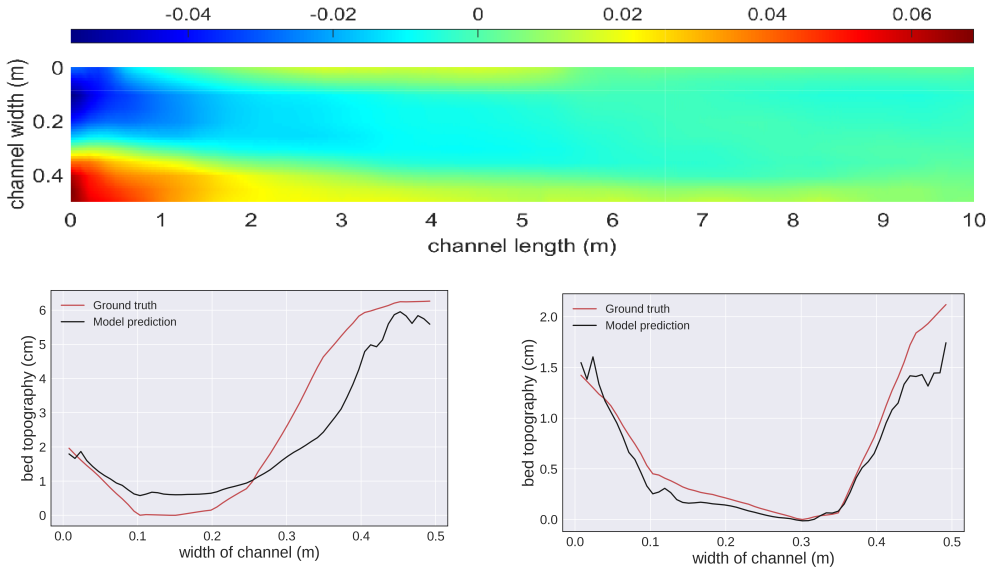


Figure 11. (**Up**): bed topography based on simulation using Iber, (**down**): bathymetry comparison of the Iber and neural network model. The red line is the Iber topography for a cross-section at $x = 1.0$ m (**down-left**) and $x = 5.5$ m (**down-right**) from the flume inlet, and the black line represents the neural network model's prediction.

3.2.3. Model's Predictions Based on Field Data

Predicting the trained neural network model on the field data would be interesting to investigate. Thus, we performed model predictions on a river located in east-central Illinois state in the USA. The work of Lewis and Rhoads [71,72] on the confluence stream provides the velocity measurements using the acoustic Doppler velocimeter (ADV—Nortek Vectrino+) in addition to bed topography for three cross-sections. The experiments were conducted at the confluence of the Kaskaskia River and Copper Slough (KRCS), positioned at 40°04′34.1″ N, 88°20′53.9″ W, as can be seen in Figure 12. The width is about 20 m at the confluence center and approximately 10 m downstream. ADVs mounted on a top-set wading rod and placed at predetermined cross-sections within the flow were used to determine the vertical velocity field. Three cross-sections, denoted as A, C, and E (refer to Figure 12), were measured, and these measurements were utilized in our neural network model for the purpose of inferring bathymetry. The sample volume for the ADV is 0.125 cm^3, and the sampling frequency is 25 Hz; it is configured in laboratory mode [72]. ADV sampling was carried out with a downward-looking probe, 6 cm below the surface, to position the sampling volume as close to the surface as possible [71]. A total of 60–80 samples of 60 seconds in length were collected at three cross-sections during one measurement campaign. Cross-sectional ADV measurements were obtained on the same day. All field campaigns resulted in only a few centimeters of change in the water surface elevation between the beginning and the end of the measurements, which is far less than the average flow depth [72].

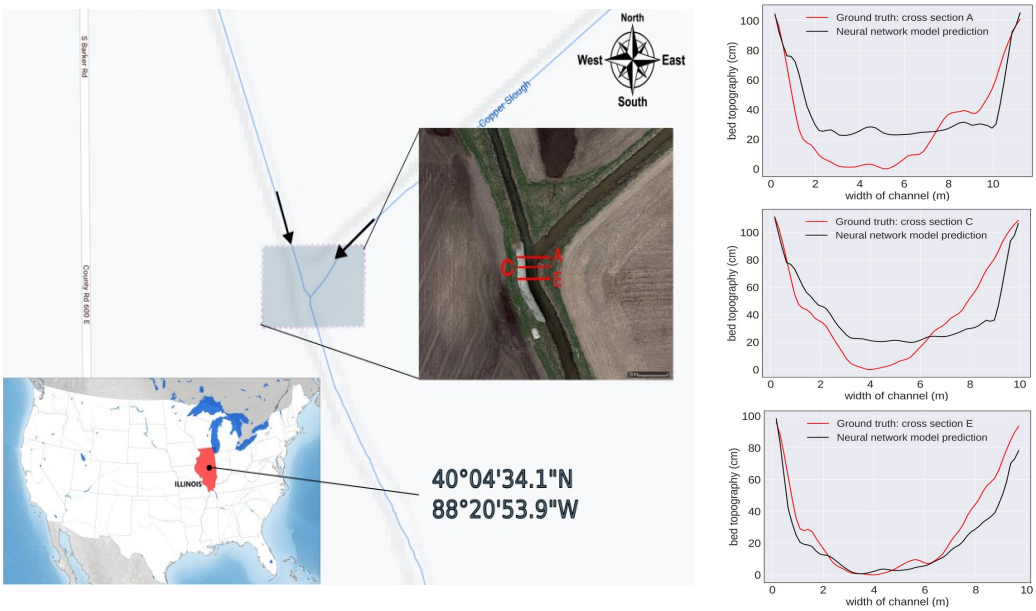


Figure 12. Left: Google Maps (2023). Modified map of the confluence of the Kaskaskia River with the Copper Slough in east-central Illinois, USA. Retrieved from https://maps.google.com/, accessed on 19 November 2023. The arrows show the flow direction. The red lines represent the cross-sections where these measurements have been conducted. **Right**: comparison of the neural network prediction vs. the field data measurements. The red and black lines represent the field measurements and neural network model's prediction, respectively.

Cross-sectional velocity fields are fed into the neural network model. Given the disparities between real-world and experimental data, our approach involves model calibration [73,74]. The neural network model, trained on an experimental dataset, establishes a mapping function denoted as f_m that connects the bathymetry and velocity fields: $y_b = f_m(v_f)$, where v_f represents the velocity field, and y_b represents the bathymetry field. Due to differences between the real-world data distribution and the laboratory dataset, this mapping function is adjusted to $y_b = \alpha f_m(v_f)$, with α as a constant calibration parame-

ter [75–77]. After analyzing one of the cross-sectional datum, we determine the optimal value for α to be 14.5, and this value is subsequently applied to the other two cross-sectional data. The bathymetry predictions for the aforementioned cross-sections based on the calibrated neural network model are compared with the field data in Figure 12's right-hand side. The ground truth (field data) is shown with a red solid line, while the prediction of the trained neural network model has been shown in black color.

The predictions' absolute errors in comparison with the measurements in cross-sections A, C, and E are 15.0, 15.6, and 6.6 cm, respectively, and the maximum relative errors are 31.3%, 18.5%, and 8.3%, respectively. However, the maximum relative error and the absolute error are still small compared to the approximately 100 cm depth of the river, and these errors could be decreased by adding a part of the field data to the training dataset, enhancing the number of training data, or using a more complicated model with more layers and training parameters.

4. Discussion

This work is concerned with imaging topography from the depth-averaged velocity field, which can be obtained more easily with a lower cost rate than the bed topography direct measurement. The network connects the velocity field and bathymetry. The U-Net is chosen for this purpose. The U-Net architecture is unique. It was first introduced for image segmentation to overcome the problem of limited training samples in the medical field—where image capturing may be expensive [45]. Its main strength is its capacity to deliver good results even with a limited amount of data, thanks to its structure. Indeed, it is able to keep traces of global and local information by analyzing the image at different scales and preserve detailed features through skip connections [27]. Skip connections from the contracting path help to retain the pieces of spatial information that were lost in the contracting path, allowing the decoder layers to more accurately detect features.

According to our tests (refer to Figure 7), error reduction requires an increase in the training dataset; use of a more complicated convolutional neural network model, such as increasing the number of feature extractions; and network layers at the cost of an increased training time. In addition, the relationship between model size and relative errors for a fixed number of training samples can be seen in Figure 7. Notably, even with the same number of training samples, different model sizes exhibit slight variations in relative errors. However, as the volume of the training data increases, the overall decrease in error remains comparable across various model sizes. When considering the smallest dataset size in Figure 7, augmenting the number of training parameters results in a corresponding increase in the relative error. However, this trend reverses when dealing with larger dataset sizes. Intriguingly, the model containing 2.3 m training parameters achieves the best relative error of 0.5% when trained on a 4k dataset. Nevertheless, it demands substantial training time, making it less practical for certain applications. On the other hand, smaller models exhibit faster training times while still delivering reasonably accurate results, which may be sufficient for the given requirements.

In Figure 9, the lower row displays the magnitudes of errors produced by the model when applied to flume experiments. These errors are relatively small in size, and their distribution across the test dataset is depicted in Figure 8b, revealing that the mean value of the errors is very close to zero. The reason for this favorable performance and relatively small error in Figure 9 can be attributed to the model's application on the test dataset. Notably, the test dataset represents a portion of the same dataset on which the model was originally trained. Consequently, this effective performance on the test data indicates that the model has been well trained and that the hyperparameters have been appropriately selected, allowing it to generalize well and produce accurate predictions when applied to the flume experiments.

The trained convolutional neural network model used the depth-averaged velocity fields, while field scientists usually measure surface velocities, for instance, by using the particle image velocimetry method. Therefore, it was worth asking how much the error

propagates when a field scientist feeds the surface velocity fields into the trained model in this work. The experiments (which we have used in this work to train the neural network) have been performed in a straight channel (no cross-stream secondary current), near the steady state, with weak sediment transport. In these conditions, the velocity profile has been obtained by Song et al. [78]. Figure 13 shows the flow velocity profile for a uniform flow at various bed slopes $0.25 < S(\%) < 1.5$, and the discharge $30 < Q(L/s) < 130$ is measured by acoustic Doppler current profilers (ADVP). The log-law for the inner region ($y/h < 0.2$) is presented with a red dashed line, and the Coles wake law could represent the velocity profile in the outer region ($y/h > 0.2$) with a blue dot-dashed line. The black solid line is the depth-averaged velocity profile.

As one can see, by increasing the flow depth, the depth-averaged velocity deviates from the velocity profile, and the maximum relative error happens on the surface flow, with a 16.8% magnitude. In order to know how this error was propagated in the model prediction, we applied the trained convolutional neural network model to the surface velocity of the field data and gained a 31.3% maximum relative error based on Figure 12. Therefore, however, the training of the neural network model is based on the depth-averaged velocity fields, but it is possible to use the surface velocity to infer bathymetry while accepting several dozen percentages of relative error. A potential source of error originates from the lateral velocity: in the trained model, the lateral velocity is virtually zero compared to what we observed in real-world scenarios. However, we believe that this issue does not prevent us from applying the model to real-world rivers.

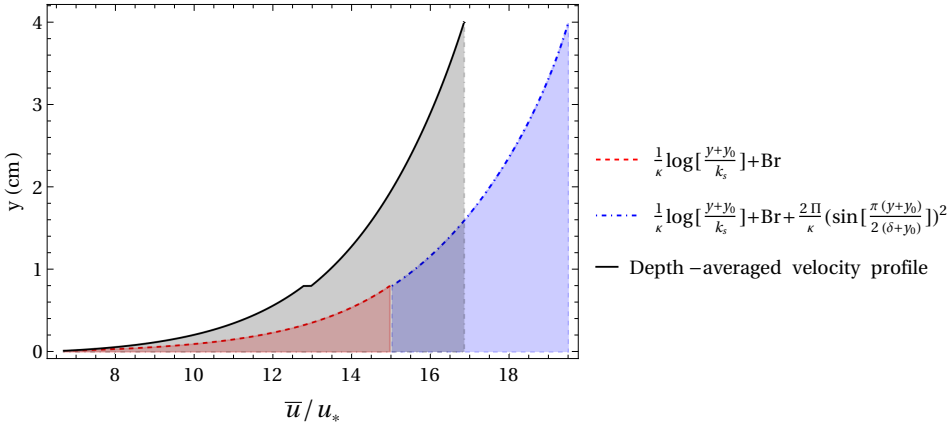


Figure 13. The red dashed line represents the log-law velocity profile in the inner region ($y/h < 0.2$) where $\bar{u}$ is the point velocity averaged for different experiments; u_* is the shear velocity; y is the distance from the top of the sediments; $y_0 = 0.2k_s$ is the viscous sublayer; k_s is the roughness height, which is equal to d_{50}; and Br=8.44 is the integration constant. The blue dot-dashed line is the velocity profile in the outer region ($y/h > 0.2$), where κ is the Von Karman's constant, $\Pi = 0.108$ is the wake strength parameter, and δ is the distance between the bed and the point at which the maximum velocity occurs; here, it is equal to h.

Another concern about this work was about the amount of data needed to suffice for training the neural network. A test on the dataset size in Figure 5b shows that 4133 data points were enough for this work's purpose, since the error does not significantly decrease after a 2500 dataset size However, having a 31.3% maximum relative error when the model is applied to real rivers is good enough for real-world applications. Further accuracy is possible using a more complicated model (which increases training time and needs more datapoints in order to avoid overfitting), but whether it is necessary depends on the application of the model. The proposed neural network model, trained on an experimental dataset with 4133 data, was applied to the flume experiments in Figures 9 and 10, to the numerical simulation in Figure 11, and to field measurements on a river in Illinois state, USA, in Figure 12. All comparisons show an acceptable prediction of the model, meaning that the model is trained properly and can be used for academic/industrial applications.

The closer the conditions are to those under which the dataset was collected, the more precise the predictions become. For instance, when deep learning models are applied to a subset of the experimental data in Section 3.2.1, the accuracy of predictions is noticeably higher. Conversely, if the flow conditions diverge from the conditions in which the dataset was collected, such as when applied to real-world river data in Section 3.2.3, the predictions become less accurate. In this section, the model is applied to the following range of hydraulic values (where the model's prediction relative error reached several dozen percentages): the average depth of the upstream tributaries measured 51 cm, while their average speed measured 0.31 m/s. Consequently, the model usage is recommended for the aforementioned flow conditions, e.g., shallow flows over gravel-bed conditions. This empirical observation explains why the simulation input parameters presented in Section 3.2.2 were chosen to closely match the experimental conditions used to train the neural network.

5. Conclusions

We applied the U-net with alternating convolutional layers to encode the velocity field (which was estimated here using a statistical method), followed by alternating convolutional and upsampling layers to decode them and a final convolutional layer with Leaky ReLU activation to infer the bed topography. The model contains skip connections, which improves training efficiency. This network has a skipping layer, which allows the decoding part of the network to have additional information about the encoding without losing any information. The final neural network has been established according to the tests performed in order to find out the best parameters: 143k trainable parameters with no dropout, learning rate of 10^{-2}, choosing 10^{-6} as the weight decay factor, and using the whole dataset, i.e., 4133 data points for the training. The network is summarized in Table A1.

The trained model does not require solving the shallow water equations numerically. It constructs the solution (bathymetry) by just looking at the input data (velocity fields). Overall, after studying the basic parameters and hyperparameters, and training a neural network model, we have found the best model yields a good accuracy (less than a 1% relative error for estimated bathymetry) when working on the test dataset (laboratory experiments), less than a 20% maximum relative error when applying the model to the numerical simulations (using Iber), and with a maximum 31.3% relative error by applying the model to field data (the confluence of Kaskaskia River with the Copper Slough).

This work shows the possibility of using the U-net architecture for predicting the bathymetry from the velocity field and provides a user-friendly tool for whoever has the velocity field and is interested in deducing the bed topography. The user needs to be aware that this is not a general model for every kind of flume or river. This work is based on gravel-bed flume/river, and, therefore, the model provides accurate bathymetry for gravel-bed rivers. The strength of the proposed model lies in its trained on experimental data, and the neural network model avoids solving the partial differential equations that govern the flow and bed, providing quick access to bed topography.

Author Contributions: Conceptualization, M.K.-O.; methodology, M.K.-O.; software, M.K.-O.; validation, M.K.-O.; formal analysis, M.K.-O.; investigation, M.K.-O.; resources, M.K.-O.; data curation, M.K.-O.; writing—original draft preparation, M.K.-O.; writing—review and editing, M.K.-O. and C.A.; visualization, M.K.-O.; supervision, C.A.; project administration, C.A.; funding acquisition, C.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Swiss National Science Foundation grant number 200020_204108.

Data Availability Statement: The dataset can be found at https://rb.gy/84l4hk, accessed on 19 November 2023.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. Model Architecture

The neural network consists of a total of 12 layers, with each layer including activation, upsampling, convolution, batch normalization, and dropout operations. Table A1 shows the structure of the U-net used in all of our tests. The network weights were initialized following He et al. [64]. After conducting some tests, we found out that dropout does not aid learning by the model, so we do not use it. Instead, we utilize batch normalization. Training was performed using Adam [61] with $\beta_{1,2} = [0.5, 0.999]$ and $\epsilon = 10^{-8}$ parameter values. The learning rate was kept at a constant value during training. A 0.01 learning rate and a 100 minibatch size were used in all tests.

Table A1. Model summary.

Layer (Type)	Output Shape	Function Parameters
InputLayer	[1, 256, 64]	
Conv2-D-1	[16, 128, 32]	n_{out}/n_{in} = 2, kernel size 4 × 4, stride = 2, pad = 1
BatchNorm2-D-2	[16, 128, 32]	
Dropout2-D-3	[16, 128, 32]	p = 0, inplace = True
LeakyReLU-4	[16, 128, 32]	negative slope = 0.2, inplace = True
Conv2-D-5	[16, 64, 16]	n_{out}/n_{in} = 2, kernel size 4 × 4, stride = 2, pad = 1
BatchNorm2-D-6	[16, 64, 16]	
Dropout2-D-7	[16, 64, 16]	p = 0, inplace = True
LeakyReLU-8	[16, 64, 16]	negative slope = 0.2, inplace = True
Conv2-D-9	[32, 32, 8]	n_{out}/n_{in} = 4, kernel size 4 × 4, stride = 2, pad = 1
BatchNorm2-D-10	[32, 32, 8]	
Dropout2-D-11	[32, 32, 8]	p = 0, inplace = True
LeakyReLU-12	[32, 32, 8]	negative slope = 0.2, inplace = True
Conv2-D-13	[64, 16, 4]	n_{out}/n_{in} = 8, kernel size 4 × 4, stride = 2, pad = 1
BatchNorm2-D-14	[64, 16, 4]	
Dropout2-D-15	[64, 16, 4]	p = 0, inplace = True
LeakyReLU-16	[64, 16, 4]	negative slope = 0.2, inplace = True
Conv2-D-17	[64, 8, 2]	n_{out}/n_{in} = 8, kernel size 2 × 2, stride = 2, pad = 0
BatchNorm2-D-18	[64, 8, 2]	
Dropout2-D-19	[64, 8, 2]	p = 0, inplace = True
LeakyReLU-20	[64, 8, 2]	negative slope = 0.2, inplace = True
Conv2-D-21	[64, 4, 1]	n_{out}/n_{in} = 8, kernel size 2 × 2, stride = 2, pad = 0
BatchNorm2-D-22	[64, 4, 1]	
Dropout2-D-23	[64, 4, 1]	p = 0, inplace = True
LeakyReLU-24	[64, 4, 1]	negative slope = 0.2, inplace = True
Upsample-25	[64, 8, 2]	scale factor = 2, mode = 'bilinear'
Conv2-D-26	[64, 8, 2]	n_{out}/n_{in} = 8, kernel size 1 × 1, stride = 1, pad = 0
BatchNorm2-D-27	[64, 8, 2]	
Dropout2-D-28	[64, 8, 2]	p = 0, inplace = True
ReLU-29	[64, 8, 2]	inplace = True
Upsample-30	[128, 16, 4]	scale factor = 2, mode = 'bilinear'
Conv2-D-31	[64, 16, 4]	n_{out}/n_{in} = 8, kernel size 1 × 1, stride = 1, pad = 0
BatchNorm2-D-32	[64, 16, 4]	
Dropout2-D-33	[64, 16, 4]	p = 0, inplace = True
ReLU-34	[64, 16, 4]	inplace = True
Upsample-35	[128, 32, 8]	scale factor = 2, mode = 'bilinear'
Conv2-D-36	[32, 32, 8]	n_{out}/n_{in} = 4, kernel size 3 × 3, stride = 1, pad = 1
BatchNorm2-D-37	[32, 32, 8]	
Dropout2-D-38	[32, 32, 8]	p = 0, inplace = True
ReLU-39	[32, 32, 8]	inplace = True
Upsample-40	[64, 64, 16]	scale factor = 2, mode = 'bilinear'
Conv2-D-41	[16, 64, 16]	n_{out}/n_{in} = 2, kernel size 3 × 3, stride = 1, pad = 1
BatchNorm2-D-42	[16, 64, 16]	

Table A1. *Cont.*

Layer (Type)	Output Shape	Function Parameters
Dropout2-D-43	[16, 64, 16]	p = 0, inplace = True
ReLU-44	[16, 64, 16]	inplace = True
Upsample-45	[32, 128, 32]	scale factor = 2, mode = 'bilinear'
Conv2-D-46	[16, 128, 32]	n_{out}/n_{in} = 2, kernel size 3 × 3, stride = 1, pad = 1
BatchNorm2-D-47	[16, 128, 32]	
Dropout2-D-48	[16, 128, 32]	p = 0, inplace = True
ReLU-49	[16, 128, 32]	inplace = True
Upsample-50	[32, 256, 64]	scale factor = 2, mode = 'bilinear'
Conv2-D-51	[1, 256, 64]	n_{out}/n_{in} = 1, kernel size 3 × 3, stride = 1, pad = 1
Dropout2-D-52	[1, 256, 64]	p = 0, inplace = True

Appendix B. Tsallis Entropy-Based Method

Tsallis introduced the generalization of Boltzmann–Gibbs–Shannon (known as Shannon) statistics [79]. According to the Tsallis entropy theory, the 2D velocity equation is expressed as follows [33,34]

$$\frac{u}{u_{max}} = \frac{2}{G}\left[G\frac{y}{h+D}\exp\left(1-\frac{y}{h+D}\right)+\frac{(4-G)^2}{16}\right]^{1/2}-\left(\frac{4-G}{2G}\right) \quad \text{(A1)}$$

where the entropic parameter is denoted as G. Here, $D = 0$ (meaning that the maximum velocity happens on the surface flow) because the flume is considered wide $w/h > 3.5$, and, on the other hand, Song's experiments emphasize this hypothesis [43]. The definition of G is

$$\frac{u_m}{u_{max}} = \frac{12+G}{24} \quad \text{(A2)}$$

The mean velocity in a cross-section u_m is known based on reckoning the flow discharge and the section area, while the u_{max} is determined iteratively by the algorithm expressed in Section 2.2.1, by knowing the maximum velocity happening on the flow surface, assuming the maximum velocity in the first iteration, and correcting the maximum velocity iteratively via calculation of the flow discharge—using the estimated velocity profile and comparing it with the ground truth.

References

1. Marcus, W.A. Mapping of stream microhabitats with high spatial resolution hyperspectral imagery. *J. Geogr. Syst.* **2002**, *4*, 113–126. [CrossRef]
2. Wozencraft, J.; Millar, D. Airborne lidar and integrated technologies for coastal mapping and nautical charting. *Mar. Technol. Soc. J.* **2005**, *39*, 27–35. [CrossRef]
3. Hilldale, R.C.; Raff, D. Assessing the ability of airborne LiDAR to map river bathymetry. *Earth Surf. Process. Landforms* **2008**, *33*, 773–783. [CrossRef]
4. Zaron, E.D. Recent developments in bottom topography mapping using inverse methods. In *Data Assimilation for Atmospheric, Oceanic and Hydrologic Applications (Vol. III)*; Springer: Berlin/Heidelberg, Germany, 2017, pp. 241–258.
5. Durand, M.; Andreadis, K.M.; Alsdorf, D.E.; Lettenmaier, D.P.; Moller, D.; Wilson, M. Estimation of bathymetric depth and slope from data assimilation of swath altimetry into a hydrodynamic model. *Geophys. Res. Lett.* **2008**, *35*. [CrossRef]
6. Simeonov, J.A.; Holland, K.T.; Anderson, S.P. River discharge and bathymetry estimation from inversion of surface currents and water surface elevation observations. *J. Atmos. Ocean. Technol.* **2019**, *36*, 69–86. [CrossRef]
7. Emery, L.; Smith, R.; McNeal, D.; Hughes, B.; Swick, L.W.; MacMahan, J. Autonomous collection of river parameters using drifting buoys. In Proceedings of the OCEANS 2010 MTS/IEEE SEATTLE, Seattle, WA, USA, 20–23 September 2010; pp. 1–7.
8. Almeida, T.G.; Walker, D.T.; Warnock, A.M. Estimating river bathymetry from surface velocity observations using variational inverse modeling. *J. Atmos. Ocean. Technol.* **2018**, *35*, 21–34. [CrossRef]
9. Smith, J.D.; McLean, S. A model for flow in meandering streams. *Water Resour. Res.* **1984**, *20*, 1301–1315. [CrossRef]
10. Wilson, G.; Özkan-Haller, H.T. Ensemble-based data assimilation for estimation of river depths. *J. Atmos. Ocean. Technol.* **2012**, *29*, 1558–1568. [CrossRef]

11. Landon, K.C.; Wilson, G.W.; Özkan-Haller, H.T.; MacMahan, J.H. Bathymetry estimation using drifter-based velocity measurements on the Kootenai River, Idaho. *J. Atmos. Ocean. Technol.* **2014**, *31*, 503–514. [CrossRef]
12. Lee, J.; Ghorbanidehno, H.; Farthing, M.W.; Hesser, T.J.; Darve, E.F.; Kitanidis, P.K. Riverine bathymetry imaging with indirect observations. *Water Resour. Res.* **2018**, *54*, 3704–3727. [CrossRef]
13. Ghorbanidehno, H.; Lee, J.; Farthing, M.; Hesser, T.; Darve, E.F.; Kitanidis, P.K. Deep learning technique for fast inference of large-scale riverine bathymetry. *Adv. Water Resour.* **2021**, *147*, 103715. [CrossRef]
14. Honnorat, M.; Monnier, J.; Rivière, N.; Huot, É.; Le Dimet, F.X. Identification of equivalent topography in an open channel flow using Lagrangian data assimilation. *Comput. Vis. Sci.* **2010**, *13*, 111–119. [CrossRef]
15. MacMahan, J.; Brown, J.; Thornton, E. Low-cost handheld global positioning system for measuring surf-zone currents. *J. Coast. Res.* **2009**, *25*, 744–754. [CrossRef]
16. Najar, M.A.; Benshila, R.; Bennioui, Y.E.; Thoumyre, G.; Almar, R.; Bergsma, E.W.; Delvit, J.M.; Wilson, D.G. Coastal bathymetry estimation from Sentinel-2 satellite imagery: Comparing deep learning and physics-based approaches. *Remote Sens.* **2022**, *14*, 1196. [CrossRef]
17. Liu, X.; Song, Y.; Shen, C. Bathymetry Inversion using a Deep-Learning-Based Surrogate for Shallow Water Equations Solvers. *arXiv* **2022**, arXiv:2203.02821.
18. Forghani, M.; Qian, Y.; Lee, J.; Farthing, M.W.; Hesser, T.; Kitanidis, P.K.; Darve, E.F. Application of deep learning to large scale riverine flow velocity estimation. *Stoch. Environ. Res. Risk Assess.* **2021**, *35*, 1069–1088. [CrossRef]
19. Yu, S.; Ma, J. Deep learning for geophysics: Current and future trends. *Rev. Geophys.* **2021**, *59*, e2021RG000742. [CrossRef]
20. Jackson, J.E. *A User's Guide to Principal Components*; John Wiley & Sons: Hoboken, NJ, USA, 2005.
21. Bellman, R. Dynamic programming. *Science* **1966**, *153*, 34–37. [CrossRef]
22. Maximenko, N.; Hafner, J.; Niiler, P. Pathways of marine debris derived from trajectories of Lagrangian drifters. *Mar. Pollut. Bull.* **2012**, *65*, 51–62. [CrossRef]
23. Bradley, A.A.; Kruger, A.; Meselhe, E.A.; Muste, M.V. Flow measurement in streams using video imagery. *Water Resour. Res.* **2002**, *38*, 51–1. [CrossRef]
24. Lewis, Q.W.; Rhoads, B.L. Resolving two-dimensional flow structure in rivers using large-scale particle image velocimetry: An example from a stream confluence. *Water Resour. Res.* **2015**, *51*, 7977–7994. [CrossRef]
25. Biondi, F.; Addabbo, P.; Clemente, C.; Orlando, D. Measurements of surface river doppler velocities with along-track InSAR using a single antenna. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2020**, *13*, 987–997. [CrossRef]
26. Yoon, Y.; Durand, M.; Merry, C.J.; Clark, E.A.; Andreadis, K.M.; Alsdorf, D.E. Estimating river bathymetry from data assimilation of synthetic SWOT measurements. *J. Hydrol.* **2012**, *464*, 363–375. [CrossRef]
27. Ronneberger, O.; Fischer, P.; Brox, T. U-net: Convolutional networks for biomedical image segmentation. In Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention, Munich, Germany, 5–9 October 2015; pp. 234–241.
28. Drozdzal, M.; Vorontsov, E.; Chartrand, G.; Kadoury, S.; Pal, C. The importance of skip connections in biomedical image segmentation. In Proceedings of the International Workshop on Deep Learning in Medical Image Analysis, International Workshop on Large-Scale Annotation of Biomedical Data and Expert Label Synthesis, Athens, Greece, 21 October 2016; pp. 179–187.
29. Orhan, A.E.; Pitkow, X. Skip connections eliminate singularities. *arXiv* **2017**, arXiv:1701.09175.
30. Dhont, B.E.M. *Sediment Pulses in a Gravel-Bed Flume with Alternate Bars*; Technical report; EPFL: Lausanne, Switzerland, 2017.
31. Griffiths, G.A. Sediment translation waves in braided gravel-bed rivers. *J. Hydraul. Eng.* **1993**, *119*, 924–937. [CrossRef]
32. Venditti, J.; Nelson, P.; Minear, J.; Wooster, J.; Dietrich, W. Alternate bar response to sediment supply termination. *J. Geophys. Res. Earth Surf.* **2012**, *117*. [CrossRef]
33. Singh, V.P. *Introduction to Tsallis Entropy Theory in Water Engineering*; CRC Press: Boca Raton, FL, USA, 2016.
34. Vyas, J.K.; Perumal, M.; Moramarco, T. Discharge estimation using Tsallis and Shannon entropy theory in natural channels. *Water* **2020**, *12*, 1786. [CrossRef]
35. Jaynes, E.T. Information theory and statistical mechanics. *Phys. Rev.* **1957**, *106*, 620. [CrossRef]
36. Singh, V.P.; Yang, C.T.; Deng, Z. Downstream hydraulic geometry relations: 1. Theoretical development. *Water Resour. Res.* **2003**, *39* . [CrossRef]
37. Bechle, A.J.; Wu, C.H. An entropy-based surface velocity method for estuarine discharge measurement. *Water Resour. Res.* **2014**, *50*, 6106–6128. [CrossRef]
38. Chiu, C.L. Entropy and probability concepts in hydraulics. *J. Hydraul. Eng.* **1987**, *113*, 583–599. [CrossRef]
39. Chiu, C.L. Entropy and 2-D velocity distribution in open channels. *J. Hydraul. Eng.* **1988**, *114*, 738–756. [CrossRef]
40. Chiu, C.L. Velocity distribution in open channel flow. *J. Hydraul. Eng.* **1989**, *115*, 576–594. [CrossRef]
41. Kumbhakar, M.; Ghoshal, K.; Singh, V.P. Two-dimensional distribution of streamwise velocity in open channel flow using maximum entropy principle: Incorporation of additional constraints based on conservation laws. *Comput. Methods Appl. Mech. Eng.* **2020**, *361*, 112738. [CrossRef]
42. Moramarco, T.; Saltalippi, C.; Singh, V.P. Estimation of mean velocity in natural channels based on Chiu's velocity distribution equation. *J. Hydrol. Eng.* **2004**, *9*, 42–50. [CrossRef]
43. Song, T.; Graf, W.H. Velocity and Turbulence Distribution in Unsteady Open-Channel Flows. *J. Hydraul. Eng.* **1996**, *122*, 141–154. [CrossRef]

44. Kohl, S.; Romera-Paredes, B.; Meyer, C.; De Fauw, J.; Ledsam, J.R.; Maier-Hein, K.; Eslami, S.; Jimenez Rezende, D.; Ronneberger, O. A probabilistic u-net for segmentation of ambiguous images. *Adv. Neural Inf. Process. Syst.* **2018**, *31*.
45. Siddique, N.; Paheding, S.; Elkin, C.P.; Devabhaktuni, V. U-net and its variants for medical image segmentation: A review of theory and applications. *IEEE Access* **2021**, *9*, 82031–82057. [CrossRef]
46. Rafique, M.U.; Zhu, J.; Jacobs, N. Automatic segmentation of sinkholes using a convolutional neural network. *Earth Space Sci.* **2022**, *9*, e2021EA002195. [CrossRef]
47. Sudhan, M.; Sinthuja, M.; Pravinth Raja, S.; Amutharaj, J.; Charlyn Pushpa Latha, G.; Sheeba Rachel, S.; Anitha, T.; Rajendran, T.; Waji, Y.A. Segmentation and classification of glaucoma using U-net with deep learning model. *J. Healthc. Eng.* **2022**, *2022*. [CrossRef]
48. Yan, C.; Fan, X.; Fan, J.; Wang, N. Improved U-Net remote sensing classification algorithm based on Multi-Feature Fusion Perception. *Remote Sens.* **2022**, *14*, 1118. [CrossRef]
49. Windheuser, L.; Karanjit, R.; Pally, R.; Samadi, S.; Hubig, N. An end-to-end flood stage prediction system using deep neural networks. *Earth Space Sci.* **2023**, *10*, e2022EA002385. [CrossRef]
50. Thuerey, N.; Weißenow, K.; Prantl, L.; Hu, X. Deep learning methods for Reynolds-averaged Navier–Stokes simulations of airfoil flows. *AIAA J.* **2020**, *58*, 25–36. [CrossRef]
51. Yao, W.; Zeng, Z.; Lian, C.; Tang, H. Pixel-wise regression using U-Net and its application on pansharpening. *Neurocomputing* **2018**, *312*, 364–371. [CrossRef]
52. Jiang, Z.; Tahmasebi, P.; Mao, Z. Deep residual U-net convolution neural networks with autoregressive strategy for fluid flow predictions in large-scale geosystems. *Adv. Water Resour.* **2021**, *150*, 103878. [CrossRef]
53. Beheshti, N.; Johnsson, L. Squeeze u-net: A memory and energy efficient image segmentation network. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, Seattle, WA, USA, 14–19 June 2020; pp. 364–365.
54. Huet-Dastarac, M.; Nguyen, D.; Jiang, S.; Lee, J.; Montero, A.B. Can input reconstruction be used to directly estimate uncertainty of a regression U-Net model?–Application to proton therapy dose prediction for head and neck cancer patients. *arXiv* **2023**, arXiv:2310.19686.
55. Alom, M.Z.; Hasan, M.; Yakopcic, C.; Taha, T.M.; Asari, V.K. Recurrent residual convolutional neural network based on u-net (r2u-net) for medical image segmentation. *arXiv* **2018**, arXiv:1802.06955.
56. Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. Pytorch: An imperative style, high-performance deep learning library. In *Proceedings of the Advances in Neural Information Processing Systems 32 (NeurIPS 2019), Vancouver, BC, Canada, 8–14 December 2019*; Curran Associates, Inc.: Red Hook, NY, USA, 2019; Volume 32.
57. Guo, C.; Szemenyei, M.; Hu, Y.; Wang, W.; Zhou, W.; Yi, Y. Channel attention residual u-net for retinal vessel segmentation. In Proceedings of the ICASSP 2021–2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Toronto, ON, Canada, 6–11 June 2021; pp. 1185–1189.
58. Shekhar, S.; Bansode, A.; Salim, A. A comparative study of hyper-parameter optimization tools. In Proceedings of the 2021 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE), Brisbane, Australia, 8–10 December 2021; pp. 1–6.
59. Ghosh, S.; Singh, A.; Kumar, S. BBBC-U-Net: Optimizing U-Net for automated plant phenotyping using big bang big crunch global optimization algorithm. *Int. J. Inf. Technol.* **2023**, *15*, 4375–4387. [CrossRef]
60. Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*; MIT Press: Cambridge, MA, USA, 2016. Available online: http://www.deeplearningbook.org (accessed on 19 November 2023).
61. Kingma, D.P.; Ba, J. Adam: A method for stochastic optimization. *arXiv* **2014**, arXiv:1412.6980.
62. Santurkar, S.; Tsipras, D.; Ilyas, A.; Madry, A. How does batch normalization help optimization? In *Proceedings of the Advances in Neural Information Processing Systems 31 (NeurIPS 2018), Montréal, QC, Canada, 3–8 December 2018*; Curran Associates, Inc.: Red Hook, NY, USA, 2018; Volume 31.
63. Ioffe, S.; Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Proceedings of the International Conference on Machine Learning, PMLR, Lille, France, 7–9 July 2015; pp. 448–456.
64. He, K.; Zhang, X.; Ren, S.; Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In Proceedings of the IEEE International Conference on Computer Vision, Santiago, Chile, 7–13 December 2015; pp. 1026–1034.
65. Glorot, X.; Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, Chia Laguna Resort, Italy, 13–15 May 2010; pp. 249–256.
66. Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; Salakhutdinov, R. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* **2014**, *15*, 1929–1958.
67. Loshchilov, I.; Hutter, F. Decoupled weight decay regularization. *arXiv* **2017**, arXiv:1711.05101.
68. Loshchilov, I.; Hutter, F. Fixing Weight Decay Regularization in Adam. 2018. Available online: https://openreview.net/forum?id=rk6qdGgCZ (accessed on 19 November 2023).
69. Zhang, C.; Bengio, S.; Hardt, M.; Recht, B.; Vinyals, O. Understanding deep learning (still) requires rethinking generalization. *Commun. ACM* **2021**, *64*, 107–115. [CrossRef]
70. Bladé, E.; Cea, L.; Corestein, G.; Escolano, E.; Puertas, J.; Vázquez-Cendón, E.; Dolz, J.; Coll, A. Iber: Herramienta de simulación numérica del flujo en ríos. *Rev. Int. Métodos Numéricos Cálculo Diseño Ing.* **2014**, *30*, 1–10. [CrossRef]

71. Lewis, Q.W.; Rhoads, B.L. LSPIV measurements of two-dimensional flow structure in streams using small unmanned aerial systems: 1. Accuracy assessment based on comparison with stationary camera platforms and in-stream velocity measurements. *Water Resour. Res.* **2018**, *54*, 8000–8018. [CrossRef]
72. Lewis, Q.W.; Rhoads, B.L. LSPIV measurements of two-dimensional flow structure in streams using small unmanned aerial systems: 2. Hydrodynamic mapping at river confluences. *Water Resour. Res.* **2018**, *54*, 7981–7999. [CrossRef]
73. Farahani, A.; Voghoei, S.; Rasheed, K.; Arabnia, H.R. A brief review of domain adaptation. In *Advances in Data Science and Information Engineering: Proceedings from ICDATA 2020 and IKE 2020, Las Vegas, NV, USA, 27–30 July 2020*; Springer: Berlin/Heidelberg, Germany, 2021; pp. 877–894.
74. Tsai, W.P.; Feng, D.; Pan, M.; Beck, H.; Lawson, K.; Yang, Y.; Liu, J.; Shen, C. From calibration to parameter learning: Harnessing the scaling effects of big data in geoscientific modeling. *Nat. Commun.* **2021**, *12*, 5988. [CrossRef] [PubMed]
75. Yang, Y.; Pan, M.; Beck, H.E.; Fisher, C.K.; Beighley, R.E.; Kao, S.C.; Hong, Y.; Wood, E.F. In quest of calibration density and consistency in hydrologic modeling: Distributed parameter calibration against streamflow characteristics. *Water Resour. Res.* **2019**, *55*, 7784–7803. [CrossRef]
76. Gao, H.; Birkel, C.; Hrachowitz, M.; Tetzlaff, D.; Soulsby, C.; Savenije, H.H. A simple topography-driven and calibration-free runoff generation module. *Hydrol. Earth Syst. Sci.* **2019**, *23*, 787–809. [CrossRef]
77. Acuña, G.J.; Ávila, H.; Canales, F.A. River model calibration based on design of experiments theory. A case study: Meta River, Colombia. *Water* **2019**, *11*, 1382. [CrossRef]
78. Song, T.; Graf, W.; Lemmin, U. Uniform flow in open channels with movable gravel bed. *J. Hydraul. Res.* **1994**, *32*, 861–876. [CrossRef]
79. Tsallis, C. Possible generalization of Boltzmann-Gibbs statistics. *J. Stat. Phys.* **1988**, *52*, 479–487. [CrossRef]