Towards AI for approximating hydrodynamic simulations as a 2D segmentation task

Lydia Bryan-Smith1 and Nina Dethlefs2
1,2University of Hull

1 Abstract

Traditional predictive simulations and remote sensing techniques for forecasting floods are based on fixed and spatially restricted physics-based models. These models are computationally expensive and can take many hours to run, resulting in predictions made based on outdated data. They are also spatially fixed, and unable to scale to unknown areas. By modelling the task as an image segmentation problem, an alternative approach using artificial intelligence to approximate the parameters of a physics-based model in 2D is demonstrated, enabling rapid predictions to be made in real-time.

2 Introduction

Predicting floods and other events caused by extreme weather relies on computational simulations using techniques such as cellular automata to predict them in advance [1]. These events can have severe impacts, such as destruction of property and injury and/or loss of life, making their prediction in a timely manner essential to avoid humanitarian disasters [2].

Physics-based simulations are used in a variety of contexts, including hydrology [3–5], climate [1], and others [6, 7] - but while accurate are computationally expensive and time consuming to run, resulting in predictions made using outdated data. This makes them unsuited for real-time estimation of hazardous phenomena in situations in which conditions are unpredictable.

AI has previously been used for predicting and monitoring such events in the form of simple ANNs [8], sequence to sequence (LSTM) [9], image classification [10–12], and more [13], but these approaches are limited to a single geographical point at a time (e.g. the flow rate of a river at a single point) - leaving characterisation of the flood itself to a hydrodynamic model (as explained in section 3.1 [9]).

In this paper, we present a feasibility study for approximating the parameters of a physics-based hydrological simulation [4] with an AI model. We achieve this by modelling the problem as an image semantic segmentation task. Our model, given an input of T time steps of input data that would nominally be fed to a physics-based model, predicts the output this computational model would have produced in real time.

While for our approach we utilise water depth prediction from rainfall radar data [14] and a heightmap [15], we aim that our approach be applicable to other domains. We make the following contributions in this paper:

- A state-of-the-art model architecture for predicting water depth in 2D
- The outline of a system could be used in harmony with existing hydrodynamic models to map floods in real time
- An illustration of improvements made from a baseline DeepLabV3+ model and their effectiveness in achieving accurate predictions, and the challenges that still remain

3 Related works

3.1 Computational simulations

Hydrological simulations to map and simulate the flow of water are available using a variety of algorithms. Models are either particle based [16] or cell based [4]. Models with 3 dimensions are the most computationally expensive, so are not used in this project. These are followed by models in 2D such as CAESAR-Lisflood [4] which have a lower computational cost.

Unfortunately, these models are still rather computationally expensive. Attempts have been made to reduce the computational requirements of these models. One option is to utilise GPU acceleration [5] or cellular automata [4], but while this does reduce the computational complexity it is not enough to run the model in real time over a large area.

Another avenue that has been explored to reduce this cost is simplifying the underlying algorithm. Such simplified algorithms [17, 18] are an order of magnitude less resource intensive, so as a result can process wider areas at once than e.g. 2D models implementing shallow water equations, but are less accurate [19].

At the other extreme, large climate simulations run by governments on high performance computers have the capability to simulate many parameters at once for large areas [1, 20], but are often commercial.
and access to data is limited, aside from the clear computational cost barrier.

### 3.2 Learning Models

With the advent of Deep Learning, it has been used in a variety of contexts to predict floods. The simplest of these approaches is to predict water depth at a single point in a river a given amount of time in advance [9], or multiple points across a river basin [21]. This is only a proxy for predicting water depth across a wider area though, with a physics based model (section 3.1) required for this purpose.

Covering wider areas with AI has been tackled in a number of ways. Hybrid models involve using AI to make some prediction, and then hand off to a physics-based model [5, 22, 23]. Another option is to make and/or predict indirect measurements (e.g. the state of a municipal drainage system), and use these to predict the actual state of the world [24]. All of these options either have the same computational expense limitations as the hydrological models they call or make predictions for single geographical points, limiting generalisability.

Alternatively, data from satellites or automated drones can be used to estimate the extent of a flood [25–27]. These methods can produce accurate maps of the extent of a flooding event, but are either expensive and high effort (drones) or low frequency and sometimes limited by clouds (satellites).

### 4 Data

Before we can train a model (see section 5.1), a ground truth label upon which to train it must first be calculated. In our project, we achieve this by running the hydrological simulation model CAESAR-Lisflood [4] - specifically the C++ implementation HAIL-CAESAR [28], which we applied minor I/O format modifications to. The inputs to this model are geographically centred around the Humber estuary in the United Kingdom (as data from the UK was readily available, and the nature of the area is already known), though the model is designed to be trained on data from any location. These inputs are twofold:

1. **Rainfall radar data**: Every 5 minutes from 2006-01-01 to 2020-10-02, sized $105 \times 174$. Total timesteps: 1.48M [14].
2. **Heightmap**: Ordnance survey terrain 50 dataset [15]

The hydrological simulation model simulates water flow using simplified 2D shallow water equations, and outputs an absolute water depth map for the simulated area at the same time frequency as the input rainfall radar data, measured in metres. This map is then binarised into 2 classes: “water” (1) and “not water” (0) with a threshold of 0.1m. We calculate a ratio of 22.1%:77.9% (water:not water) between the 2 classes. This is the ground truth label used to train the model, as shown in figure 1.

Data is stored in the tensorflow TFRecord format to optimise read performance. 4000 samples are stored per file, totalling 371 files.

### 5 Approach

#### 5.1 Semantic Segmentation

Image semantic segmentation models perform the task of pixel-wise classification on some input image, segmenting it into logically distinct categories. For example, an image of a typical urban street in the form of an $128 \times 128 \times 3$ channels-last matrix could be semantically segmented into categories such as **building**, **road**, and **person**, with the output in the form $128 \times 128 \times N$, where $N$ is the number of classes to segment the input image into.

This technique has been adapted to make 2D predictions rainfall in the future, using rainfall radar data from a few hours prior [14, 29, 30]. While
rainfall predictions are valuable, this is not the only source of flooding [31] and are of limited use for flooding prediction without being coupled to a hydrological simulation.

In this project, ‘image’ semantic segmentation is used to approximate water depth from rainfall radar data in 2D. Multiple semantic segmentation model architectures have been developed in the field, and such model architectures naturally lend themselves not only to their originally intended purpose (segmenting images), but also to our problem too. The earliest attempt the authors could find at a model architecture for the task though is U-Net, which consists of a more full encoder-decoder autoencoder connected with skip connections [33]. SegNet is similar to U-Net, but improves on it by reducing memory usage (specifically by pooling directly before skip connections) [33]. PSPNet is of shape $[t, c]$ and the heightmap, as in section 4 - these are combined so that the heightmap becomes a channel of $R_{t,c}$. This has the practical effect of reducing 96 channels to just 8, significantly reducing the VRAM required - and is represented visually in figure 2. As an addendum, the ground truth label the model learns from can be represented thus:

$W_t^r[x, y] = \left( \sum_{i=0}^{c} W_{t+\text{offset}}[x - 1 : x + 1, y - 1 : y + 1] \right) \geq 1 \tag{2}$

where offset is an offset to the timestep as described in section 4 and figure 2, and square-bracket notation $[x_i : y_i]$ indicates a submatrix. This effectively causes any cell of $W_t$ with a value of 1 and all 8 neighbouring cells equal to 0 to be set to 0, removing isolated pixels.

Hence, intuitively the input to the model $R_{t,c}$ is of shape $[\text{batch}, \text{height}, \text{width}, \text{channel}]$, and the ground truth $W_t^r$ is of shape $[\text{batch}, \text{height}, \text{width}]$, or $[\text{batch}, \text{height}, \text{width}, \text{class}]$ when one-hot encoded.

6 Experiments

Our model is based on DeepLabV3+ [36], which as explained in section 5.1 consists of an image encoder (ResNet50), followed by a pyramid architecture such that the 2D output is a ‘semantic segmentation’ with each pixel being the binary classes water/no water. In other words, a per-pixel probability of each class - which is identical in shape to the ground truth labels ($W_t^r$) used for training (all one-hot encoded). To decode the one-hot encoded output, for each pixel the highest value is taken to be the predicted class (e.g. $P_{x,y} = f(P_{x,y,c})$, if c is the class) - as in $P[x,y] = i \text{ of } \max[P[x,y]]$, where i is the class index (i.e. in this case i ∈ \{0, 1\}).

The input to this model is the rainfall radar data and the heightmap, as in section 4 - these are combined so that the heightmap becomes a channel of $R_{t,c}$. Our model has following hyperparameters:

- **Total parameters:** 11.87M
Figure 3. Accuracy and Dice loss plots for the models we trained. Overfitting is observed if models are trained for too long. All adjustments (see also table 1) show improvements apart from log-cosh, which was dropped.

- **Learning rate:** 0.00001
- **Batch size:** 32
- **Encoder:** ResNet50 [38]
- **Loss function:** Additive cross-entropy and Dice loss (i.e. \( \text{loss} = \text{cel} + \text{dice} \))
- **Upscaling:** We adjusted to the original DeepLabV3+ model [37] to scale the input from 128 × 128 to 256 × 256 using nearest interpolation
- **Offset:** From rainfall radar data, water depth 5 minutes in the future is predicted.

We chose these hyperparameters based on a series of experimental comparisons. We trained our models on various Nvidia GPUs, subject to availability: Nvidia (A40), Tesla (K40m, P100), GeForce (2060, 3060). Models trained for a total of 25 epochs, and we picked the checkpoint with the highest validation accuracy. We used an 80% - 20% training-validation data split, with files being randomly allocated to each using the fisher-yates shuffling algorithm [39]. All models were trained with the same files in the same partition of the split.

With these settings, our model took a total of 3 days 7 hours to train, and 36 seconds to make a prediction (processing an entire batch at a time). The source 2D hydrological model that generated the water depth label data took 14 days 8 hours to complete. These include setup and teardown, such as loading system libraries.

The metrics from the models we trained are presented in table 1. Starting from an unmodified baseline DeepLabV3+ model, we iteratively improved the model to function more effectively on our task. We measured a number of metrics including accuracy (‘acc’), dice coefficient (‘dice coef’), and intersection-over-union (‘mean iou’). We show plots of the former two in figure 3. Performance gains are obtained relatively early in the training process, with further epochs leading to overfitting.

### 6.1 Discussion

The biggest issue we faced with our learning task was the output resolution of the semantic segmentation model - especially on the boundary between classes. Adding remove isolated() as described in section 4 (+1.3% val acc.), changing our loss function from simple cross-entropy-loss to include dice loss additively (+0.4%), and upscaling the input (x2; +6.8%) all improved validation accuracy (shown in brackets). We also experimented with using a loss function of \( \text{loss} = \text{cel} + \log(\cosh(\text{dice})) \) as [40] suggests that \( \log(\cosh(\text{dice})) \) is more tractable than simply \( \text{dice} \), but we found it to reduce our validation accuracy and all other metrics (-0.8%). We reduced the learning rate from the default of 0.001 to 0.00001, as we observed this reduced instability in validation accuracy and the dice coefficient - though this reduced our accuracy slightly (-0.2%).

The dataset is also very large (about 1.4M samples), further supporting the reduction in learning rate.

With this in mind, we pick the hyperparameters of the model with upscaling but not log-cosh. We plot predictions from each model alongside ground truth in figure 4. The adjusted model is shown to make predictions at a higher fidelity than the initial baseline, effectively matching the patterns in the ground truth by making predictions based on the input rainfall and heightmap information.

A key limitation in designing the model was visual fidelity and GPU video memory (VRAM) as described in section 5.2. Much of the design was done before the Nvidia A40 mentioned above was available (which has 48GB VRAM), limiting our model design choices to those that fit within a 16 GB VRAM envelope. With additional computational resources now available, the visual fidelity may potentially be further improved at the expense of more VRAM through additional upscaling - perhaps by x8 compared to the x2 used in the models presented here.

Another observation we make is of significant variation in validation accuracy during the training process. This is emphasised if we calculate the standard deviation of the metrics (excluding the first five epochs) as reported in table 2. We suggest that this is due to the complexity of the task and the predicted output. The complexity of the task could be reduced by adjusting the source hydrological simulation used. For example, this could include filling in holes in the heightmap, or choosing a different hydrological simulation model that...
Table 1. Metrics for our DeepLabV3+-based binarised water depth prediction models. For each model variant, the model was trained for 25 epochs, and then the epoch with the highest validation accuracy was chosen for display here. We iteratively improved on a baseline DeepLabV3+ model [36] by removing positive pixels surrounded by negative ones (ri), reducing the learning rate (lr=0.00001), including dice loss additively (dice loss), upscaling the inputs and outputs (upscaling), and adding log(cosh(dice loss)) (log-cosh) - though the latter adjustment slightly reduced performance, so was discarded.

Figure 4. Training predictions by each of the models described in table 1, along with the ground truth prediction (left) and the processed rainfall data that is used as input (1 from the left). When reading digitally, we recommend zooming in to observe the at times subtle differences between the different predictions.

We observe a general negative trend in model validation performance the more epochs it is trained for. This suggests that our model is overfitting if trained for too long. We also note that, as mentioned in section 4, our dataset is somewhat unbalanced. We used additive dice to our loss function to counteract this, but exploration of other techniques like weighted and shape-aware loss functions for example would be worthwhile.

Our experiments demonstrate the feasibility of our approach to predict floods in two dimensions. The accuracy is limited by the hydrological simulation the model is trained on, but with some adjustments to the hydrological simulation as explained above our approach could effectively and rapidly make direct predictions of water depth. We anticipate that, after proper characterisation and analysis, the model may be able to make accurate predictions up to a few hours in advance after making these improvements.

Table 2. Standard deviation values for the + upscaling model chosen in section 6.1 and presented in table 1. Significant variation was observed in validation metrics.

7 Conclusion

Using the DeepLabV3+ model [36] as a base, we have demonstrated a proof of concept for a new method of directly predicting floods from rainfall radar data and a heightmap in two dimensions. We accomplish this by approximating a hydrological simulation model. While our model’s performance is limited by the nature and accuracy of the simulation chosen (given that to the best of the authors’ knowledge no real-world dataset with a sufficient temporal resolution exists to serve as training labels), our method predicts an entire area at once - avoiding the need for models to be retrained and maintained for many individual locations or being combined with an expensive hydrological simulation to make useful multidimensional predictions.

In addition, our approach makes predictions more
directly than previous approaches that rely on e.g. static camera footage [41] river levels [9] or secondary sensor networks [24].

Such advance warning is essential for minimising humanitarian risk and preventing loss of life. To develop this proof of concept further, we want to more fully characterise the model’s strengths and weaknesses - e.g. under different weather patterns. Future work could also include tuning the source hydrological simulation based on the lessons learned about the nature of the semantic segmentation task framing to improve performance, and accounting for additional variables in our simulation such as sources of flooding other than rainfall (e.g. tidal), groundwater flow, and municipal drainage systems.

We also want to investigate developing a geographically-invariant version of the model that can use e.g. a tiled approach to make predictions for larger areas without the need for retraining the model, as the current version cannot generalise to other geographical areas without retraining. A tiled approach would split a rainfall/heightmap/water depth dataset into equal squared tiles before training to make predictions, and could also increase the resolution of predictions with a suitable hydrological model as an input.

Finally, increasing the fidelity of predictions to 3 or more classes beyond binarised “water”/“no water” would also improve usefulness of predictions made.

Ultimately, a range of approaches is required to effectively address the problem of accurately forecasting floods and their extent in real time. This includes not only hydrodynamic / climatological simulations [1, 4] and our approach to approximating them with AI, also other diverse data sources such as satellite data [42], remote sensing [24], and the analysis of human-centred approaches like crowdsourcing [43] and social media [44, 45]. Different sources complement each other and can improve visibility / situational awareness.

7.1 Acknowledgments

Lydia Bryan-Smith is funded by a PhD stipend from the University of Hull. We acknowledge the VIPER high-performance computing facility of the University of Hull and its support team.

References


