Flood Area Segmentation: A Deep Learning Framework for Pixel-Level Flood Detection from Aerial Imagery

Areen Jain
Gretchen Whitney High School

Abstract

Quickly and accurately identifying flooded land from aerial imagery can help with disaster response, damage assessment, and long-term planning. This paper studies binary semantic segmentation for flood detection, where each pixel is classified as flooded or non-flooded. Two versions of the project were built: one in Python using PyTorch and torchvision, and one in R using R torch. The Python version fine-tunes a DeepLabV3 model with a ResNet-50 backbone pretrained on ImageNet. The R version uses a custom U-Net trained from scratch because the R torch workflow does not provide the same ready-to-use pretrained vision models. Both models use the same combined Binary Cross-Entropy and soft Dice loss, and both are tested on the same held-out images. The Python DeepLabV3-ResNet50 model achieves a Dice coefficient of 0.8765 and an Intersection over Union (IoU) of 0.7865. The R U-Net achieves a Dice coefficient of 0.8003 and an IoU of 0.6889. These results show that pretrained models are very helpful for flood segmentation, but they also show that a carefully built U-Net in R can still produce strong results.

Keywords: semantic segmentation; flood detection; DeepLabV3; U-Net; convolutional neural networks; transfer learning; Dice loss; aerial imagery; PyTorch; R torch

1 Introduction

1.1 Background

Floods are among the most frequent and damaging natural disasters worldwide. They can destroy infrastructure, displace communities, and make it difficult for emergency responders to reach affected areas. Because of this, it is important to map flooded regions as quickly as possible before, during, and after a flood event. Aerial and satellite imagery are useful for this task because they can cover large areas even when roads are blocked or unsafe.

Traditional flood mapping methods often use spectral indices such as the Normalized Difference Water Index (NDWI)[1], hand-designed features, or other rule-based techniques. These methods can work well in some settings, but they often need expert tuning and may struggle when lighting, vegetation, geography, or sensor conditions change. Deep learning offers a different approach by learning useful image features directly from data, which has made it successful in many remote sensing tasks including classification, object detection, and segmentation[2].

Semantic segmentation assigns a label to every pixel in an image. For flood mapping, this means separating each image into flooded and non-flooded regions. This paper builds two segmentation pipelines for flood detection: a Python implementation based on DeepLabV3-ResNet50 with transfer learning, and an R implementation based on a custom U-Net trained from scratch. The paper has three main goals. First, it fine-tunes a pretrained DeepLabV3-ResNet50 model for binary flood segmentation. Second, it builds a similar R torch workflow using a custom U-Net to test how well R can handle this type of deep learning project. Third, it evaluates both models on the same test images using Dice and IoU, so the comparison is fair.

1.2 Literature Review

Recent flood mapping research increasingly treats flood detection as a semantic segmentation problem, where each pixel is classified as flooded or non-flooded. This is useful for disaster response because pixel-level maps can estimate flood extent and identify affected areas. Earlier approaches often relied on thresholding, spectral indices such as NDWI, or hand-designed features, but these methods can be sensitive to sensor type, lighting, vegetation, and geography [1, 2]. Fully convolutional networks made dense prediction more practical by producing segmentation maps instead of only image-level labels [3].

Several recent papers have applied deep learning to flood segmentation. Bonafilia et al.

[4] introduced Sen1Floods11, a Sentinel-1 SAR benchmark for training and testing flood-mapping algorithms. Nemni et al. [5] used a fully convolutional network for rapid flood segmentation in SAR imagery, while Rahnemoonfar et al. [6] introduced FloodNet, a high-resolution UAV imagery dataset for post-flood scene understanding. These studies show that segmentation models can be used across different remote sensing sources, including SAR, satellite, and aerial imagery.

Other studies have explored ways to improve flood segmentation architectures. Konapala et al. [7] studied Sentinel-1 and Sentinel-2 data for deep learning-based flood inundation mapping, showing the value of combining sensor information. Saha et al. [8] applied a DeepLabV3+-style model for flooded-region segmentation using UAV imagery. These papers are relevant to this project because they show the importance of multi-scale features, strong segmentation backbones, and high-resolution imagery.

U-Net is also widely used for segmentation because its encoder-decoder design and skip connections preserve spatial detail [9]. This makes it useful for flood boundaries, which are often irregular. In this project, the Python model uses a pretrained DeepLabV3-ResNet50 model to take advantage of transfer learning and multi-scale context, while the R model uses a simpler custom U-Net trained from scratch because pretrained vision models are less readily available in the R torch workflow [10].

2 Data Description

The dataset comes from Kaggle (`faizalkarim/flood-area-segmentation`)[11]. It contains aerial images paired with binary flood masks. Each RGB image has a matching grayscale mask. In the mask, pixels with values above 127 are treated as flooded, while the remaining pixels are treated as non-flooded.

Image–mask pairs are found automatically by matching filenames instead of relying on hardcoded file paths. Images are resized before training. The Python model uses 256×256 images, while the R model uses 128×128 images to make training more manageable. Pixel values are scaled to $[0, 1]$. The Python model also uses ImageNet normalization because its ResNet-50 backbone was pretrained with that format:

$$\tilde{x}_c = \frac{x_c/255 - \mu_c}{\sigma_c}, \quad c \in \{R, G, B\}, \tag{1}$$

where $(\mu_R, \mu_G, \mu_B) = (0.485, 0.456, 0.406)$ and $(\sigma_R, \sigma_G, \sigma_B) = (0.229, 0.224, 0.225)$.

The dataset is split into 80% training images and 20% test images. The same test file

list is used for both the Python and R models, so differences in performance come from the models and training setup rather than from different test sets.

Table 1: Dataset Summary

Property	Detail
Input format	RGB aerial/satellite image
Label format	Grayscale binary mask, thresholded at 127
Python image size	256×256 pixels
R image size	128×128 pixels
Train / Test split	80% / 20%
Normalization	ImageNet mean and std in Python; $[0, 1]$ scaling in R
Pair discovery	Automatic filename matching

3 Theoretical Framework

3.1 Convolutional Neural Networks

A convolutional neural network (CNN) processes an image by turning it into feature maps that become more complex in deeper layers. The main operation is convolution. For an input feature map $\mathbf{f}$ and a filter bank $\mathbf{W}$, the output is

$$g_{i,j,c} = \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} \sum_{c'=0}^{C_{\text{in}}-1} W_{m,n,c',c} f_{i+m,j+n,c'} + b_c. \tag{2}$$

A useful property of convolution is that the same filter can detect a pattern no matter where it appears in the image. This matters for flood segmentation because water may appear in different parts of an aerial image.

Convolutions are usually followed by nonlinear activations. Both models use the Rectified Linear Unit (ReLU)[12]:

$$\text{ReLU}(z) = \max(0, z). \tag{3}$$

Both models also use batch normalization[13], which normalizes intermediate activations and then applies learned scale and shift values:

$$\hat{z}_i = \frac{z_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}, \qquad \tilde{z}_i = \gamma \hat{z}_i + \beta. \tag{4}$$

This helps the model train more steadily.

3.2 DeepLabV3 and Residual Learning

The Python model uses DeepLabV3 with a ResNet-50 backbone. Residual networks make it easier to train deep CNNs by adding shortcut connections[14]:

$$\mathbf{y} = \mathcal{F}(\mathbf{x}; \{\mathbf{W}_i\}) + \mathbf{x}. \tag{5}$$

The shortcut lets information and gradients pass through the network more directly, which is one reason ResNet models can be very deep.

DeepLabV3 also uses atrous, or dilated, convolution. Atrous convolution increases the receptive field without reducing the feature map resolution:

$$g_{i,j,c} = \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} \sum_{c'=0}^{C_{\text{in}}-1} W_{m,n,c',c} f_{i+rm,j+rn,c'} + b_c, \tag{6}$$

where r is the dilation rate. DeepLabV3 combines several dilation rates in its Atrous Spatial Pyramid Pooling module, which helps it recognize both small water channels and larger flooded regions[15].

3.3 U-Net Architecture

The R model uses a custom U-Net[9]. U-Net has an encoder that downsamples the image and a decoder that upsamples it back toward the original resolution. At each scale, the decoder receives information from the encoder through a skip connection:

$$\mathbf{d}_l = \text{ConvBlock}\left(\text{concat}\left[\text{TConv}(\mathbf{d}_{l+1}), \mathbf{e}_l\right]\right). \tag{7}$$

These skip connections help preserve details such as flood boundaries, which might otherwise be lost during downsampling. Each convolutional block uses two Conv-BN-ReLU operations, and the final 1×1 convolution outputs one flood logit per pixel.

3.4 Transfer Learning

Transfer learning means starting with weights learned from a large dataset and then fine-tuning the model on a smaller target task. In computer vision, early layers often learn general features such as edges, textures, and color patterns that can transfer to new image tasks[16]. The Python model benefits from an ImageNet-pretrained ResNet-50 backbone. The R U-Net starts from random weights, so it has to learn all features from the flood dataset itself. This gives the Python model an expected advantage.

3.5 Loss Function

Both models are trained with a combined DiceBCE loss. Binary Cross-Entropy (BCE) measures pixel-wise classification error:

$$\mathcal{L}_{\text{BCE}}(\hat{\mathbf{p}}, \mathbf{y}) = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log \hat{p}_i + (1-y_i)\log(1-\hat{p}_i)\right]. \tag{8}$$

BCE is useful, but flood images can be imbalanced because dry pixels often outnumber flooded pixels. To address this, Dice loss is added. The soft Dice coefficient is

$$\text{Dice}(\hat{\mathbf{p}}, \mathbf{y}) = \frac{2\sum_{i=1}^{N}\hat{p}_i y_i + \epsilon}{\sum_{i=1}^{N}\hat{p}_i + \sum_{i=1}^{N} y_i + \epsilon}. \tag{9}$$

The Dice loss is $1 - \text{Dice}$, and the full objective is

$$\mathcal{L}_{\text{DiceBCE}} = \mathcal{L}_{\text{BCE}} + \mathcal{L}_{\text{Dice}}. \tag{10}$$

This loss works well for segmentation because it combines pixel-level accuracy with overlap between the predicted and true masks[17].

3.6 Evaluation Metrics and Optimization

At test time, predicted probabilities are thresholded at $\tau = 0.5$. Dice and IoU are computed as

$$\text{Dice} = \frac{2\,\text{TP}}{2\,\text{TP} + \text{FP} + \text{FN}}, \qquad \text{IoU} = \frac{\text{TP}}{\text{TP} + \text{FP} + \text{FN}}. \tag{11}$$

Dice measures overlap between the predicted and true masks. IoU is stricter because false positives and false negatives remain in the denominator. The two metrics are related by

$$\text{Dice} = \frac{2\,\text{IoU}}{1 + \text{IoU}}, \qquad \text{IoU} = \frac{\text{Dice}}{2 - \text{Dice}}. \tag{12}$$

Both models are optimized with AdamW[18], which is based on Adam[19] but applies weight decay separately from the adaptive gradient update:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\eta}{\sqrt{\hat{\mathbf{v}}_t} + \epsilon}\hat{\mathbf{m}}_t - \eta\lambda_w\boldsymbol{\theta}_t. \tag{13}$$

AdamW is commonly used for deep learning because it adjusts learning rates for each parameter while still applying regularization to the weights.

4 Methodology

4.1 Python Pipeline: DeepLabV3-ResNet50

The Python model loads DeepLabV3-ResNet50 from torchvision with ImageNet-pretrained weights. The original multi-class classifier is replaced with a binary 1×1 convolutional output layer:

$$\text{head} : \mathbb{R}^{H' \times W' \times 256} \to \mathbb{R}^{H' \times W' \times 1}. \tag{14}$$

The output is a single-channel logit map, which is upsampled to 256×256 and passed through sigmoid to produce flood probabilities. Newly added layers are initialized before training, while the pretrained backbone weights are kept and fine-tuned.

The model is trained for 10 epochs with batch size 4, learning rate $\eta = 0.0001$, AdamW, and DiceBCE loss. After each epoch, validation IoU is computed, and the best checkpoint is saved for final testing.

4.2 R Pipeline: Custom U-Net

The R model is a custom U-Net with three encoder stages, a bottleneck, and three matching decoder stages. Its channel dimensions are shown in Table 2. The model is trained from scratch using R torch.

Table 2: Custom U-Net Channel Dimensions

Stage	**Input Channels**	**Output Channels**
Encoder Block 1	3	8
Encoder Block 2	8	16
Encoder Block 3	16	32
Bottleneck	32	64
Decoder Block 3	96	32
Decoder Block 2	48	16
Decoder Block 1	24	8
Output Conv	8	1

The R model is trained for 10 epochs with batch size 1, learning rate $\eta = 0.0001$, AdamW, and the same DiceBCE loss. Images are resized to 128×128 to reduce training cost, and image input/output is handled using the `magick` library.

Table 3: Training Configuration

Hyperparameter	Python	R
Architecture	DeepLabV3-ResNet50	Custom U-Net
Initialization	ImageNet pretrained	Random
Image size	256×256	128×128
Batch size	4	1
Epochs	10	10
Optimizer	AdamW	AdamW
Learning rate	0.0001	0.0001
Loss	DiceBCE	DiceBCE
Threshold	0.5	0.5

To keep the comparison fair, the Python pipeline writes the test image filenames to a CSV file, and the R evaluation script reads the same file. Both models are therefore evaluated on the same images using the same Dice and IoU formulas.

5 Results and Discussion

5.1 Test Set Performance

Table 4: Test Set Performance

Model	**Test Dice**	**Test IoU**
Python — DeepLabV3-ResNet50	0.8765	0.7865
R — Custom U-Net	0.8003	0.6889
Absolute difference	0.0762	0.0976

Table 4 shows that the Python DeepLabV3-ResNet50 model performs better than the R U-Net model by 7.62 percentage points in Dice and 9.76 percentage points in IoU. Using the Dice–IoU relationship in Equation 12, a Dice score of 0.8765 gives an IoU of about 0.7808, which is close to the observed 0.7865. This shows that the reported metrics are consistent with each other.

Representative segmentation outputs for both models are provided in Appendix A. Figure A1 shows an example Python DeepLabV3-ResNet50 prediction, and Figure A2

shows an example R U-Net prediction. The training and loss curves for both models are shown in Figure A3.

The Python model's higher Dice and IoU scores suggest that it more accurately captures the main flooded regions and better matches the ground-truth flood masks. The R U-Net performs slightly lower, but it still captures many of the main flooded areas, showing that a custom U-Net can produce useful segmentation results even when trained from scratch.

5.2 Explaining the Performance Gap

The performance gap is mainly caused by three factors: transfer learning, input resolution, and multi-scale processing. First, ResNet-50 has about 25 million parameters and was pretrained on ImageNet, which contains over 1.2 million images across 1,000 classes[20]. These pretrained filters already recognize useful visual patterns such as edges and textures. The R U-Net starts from random weights, so it has to learn these patterns only from the flood dataset.

Second, the Python model uses 256×256 images, while the R model uses 128×128 images. This matters because higher resolution preserves more boundary information. Narrow water channels, irregular shorelines, and small flooded regions are easier to identify when fewer details are lost during resizing.

Third, DeepLabV3's ASPP module processes features at several dilation rates. This helps the model handle both small and large flooded areas. U-Net also combines information across scales, but it does not use the same parallel multi-rate structure as DeepLabV3.

The training curves in Figure A3 provide additional context for these results by showing how each model improved over 10 epochs and whether the losses stabilized.

5.3 Practical Implications

Even though the Python model performs better, the R U-Net still gives useful results. Its Dice score of 0.8003 shows strong overlap between the predicted and true flood masks. This is especially encouraging because the R model uses no pretrained weights, lower resolution, and batch size 1.

The Python pipeline would likely be the better choice when maximum accuracy is the main goal. The R pipeline is still valuable because it shows that researchers who work mainly in R can build a working deep learning segmentation model without leaving the R environment.

6 Conclusion

This paper compared two deep learning approaches for binary flood area segmentation from aerial imagery. The Python DeepLabV3-ResNet50 model achieved a Dice coefficient of 0.8765 and an IoU of 0.7865. The R custom U-Net achieved a Dice coefficient of 0.8003 and an IoU of 0.6889. Both models performed well, but the pretrained Python model achieved higher accuracy. The difference is mainly explained by ImageNet transfer learning, higher input resolution, and DeepLabV3's multi-scale ASPP module. At the same time, the R U-Net's performance shows that a simple encoder-decoder model with skip connections can still work well for flood segmentation, even when trained from scratch.

Future work could test multispectral and SAR imagery, add attention mechanisms to improve flood boundaries, deploy the models for real-time drone imagery, and combine DeepLabV3 and U-Net predictions in an ensemble. For R torch, having more pretrained vision models available would likely improve performance and make transfer learning more practical for future projects.

7 Supplemental Materials

The complete Python code, R Code, dataset, and instructions on how to run the codes are available in the project's Github Repository for reproducibility and further exploration. The link is: https://github.com/areenjain09/flood_segmentation

8 Acknowledgments

This work would not have been possible without the guidance of Dr. Olga Korosteleva, Professor of Statistics at California State University, Long Beach, whose mentorship and deep expertise shaped both the direction of this project and my growth as a researcher. I am equally grateful to the teachers and mentors who have invested in my academic development over the years. To my family, whose encouragement and presence has been a constant source of strength throughout this journey, I owe more than words can express.

A Additional Results and Training Curves

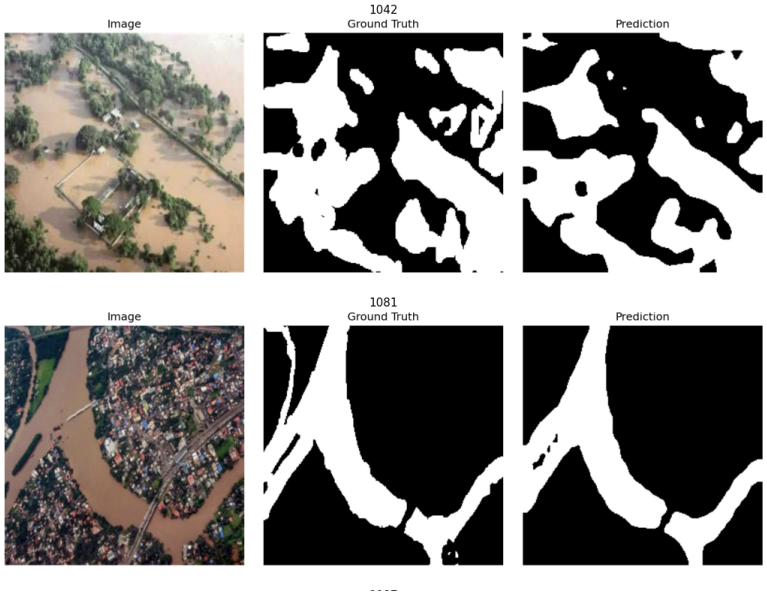

Figure A1: Example Python DeepLabV3-ResNet50 segmentation result showing the input image, ground-truth flood mask, and predicted flood mask.

Figure A2: Example R U-Net segmentation result showing the input image, ground-truth flood mask, and predicted flood mask.

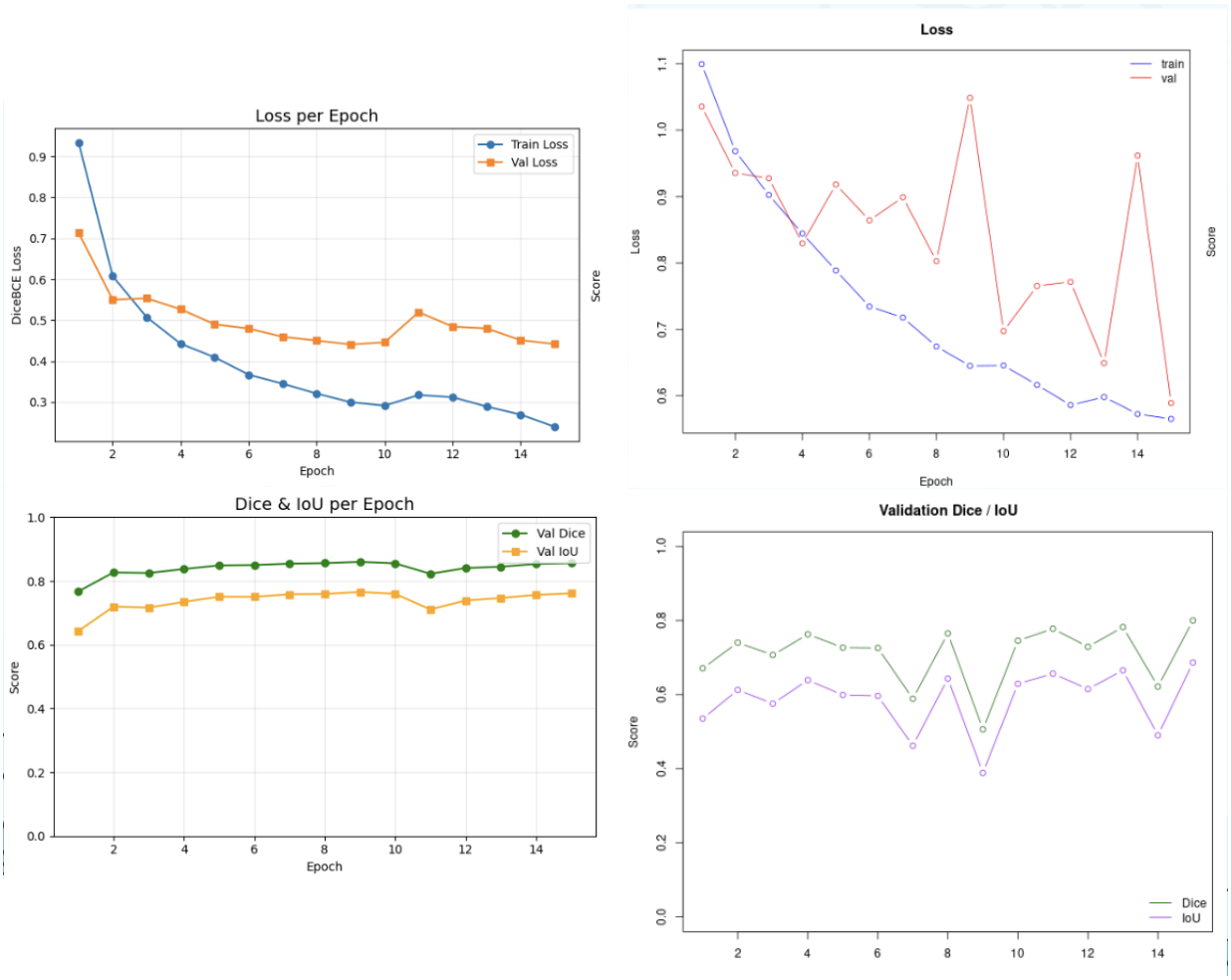

Figure A3: Training and loss curves over 10 epochs for the Python DeepLabV3-ResNet50 model and the R U-Net model.

References

[1] S. K. McFeeters, "The Use of the Normalized Difference Water Index (NDWI) in the Delineation of Open Water Features," *International Journal of Remote Sensing*, vol. 17, no. 7, pp. 1425–1432, 1996.

[2] X. X. Zhu, D. Tuia, L. Mou, G.-S. Xia, L. Zhang, F. Xu, and F. Fraundorfer, "Deep Learning in Remote Sensing: A Comprehensive Review and List of Resources," *IEEE Geoscience and Remote Sensing Magazine*, vol. 5, no. 4, pp. 8–36, 2017.

[3] J. Long, E. Shelhamer, and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 3431–3440.

[4] D. Bonafilia, B. Tellman, T. Anderson, and E. Issenberg, "Sen1Floods11: A Georeferenced Dataset to Train and Test Deep Learning Flood Algorithms for Sentinel-1," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2020.

[5] E. Nemni, J. Aylett-Bullock, S. Belabbes, and L. Bromley, "Fully Convolutional Neural Network for Rapid Flood Segmentation in Synthetic Aperture Radar Imagery," *Remote Sensing*, vol. 12, no. 16, p. 2532, 2020.

[6] M. Rahnemoonfar, T. Chowdhury, A. Sarkar, D. Varshney, M. Yari, and R. Murphy, "FloodNet: A High Resolution Aerial Imagery Dataset for Post Flood Scene Understanding," *arXiv preprint arXiv:2012.02951*, 2020.

[7] G. Konapala, S. V. Kumar, and S. K. Ahmad, "Exploring Sentinel-1 and Sentinel-2 Diversity for Flood Inundation Mapping Using Deep Learning," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 180, pp. 163–173, 2021.

[8] A. Saha, S. K. Ghosh, and M. K. Singh, "Post-Disaster Flooded Region Segmentation Using DeepLabv3+ and Unmanned Aerial Vehicle Imagery," *Natural Hazards Research*, vol. 5, no. 2, pp. 363–374, 2025.

[9] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," in *Proceedings of MICCAI 2015*, Lecture Notes in Computer Science, vol. 9351, Springer, 2015, pp. 234–241. arXiv:1505.04597.

[10] Torch Contributors, *R torch: An R Interface to PyTorch*. Available at: https://torch.mlverse.org/.

[11] F. Karim, *Flood Area Segmentation Dataset*. Kaggle, 2022. Available at: https://www.kaggle.com/datasets/faizalkarim/flood-area-segmentation.

[12] V. Nair and G. E. Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines," in *Proceedings of the 27th International Conference on Machine Learning (ICML)*, 2010, pp. 807–814.

[13] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," in *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, 2015, pp. 448–456.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[15] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking Atrous Convolution for Semantic Image Segmentation," *arXiv preprint arXiv:1706.05587*, 2017.

[16] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How Transferable Are Features in Deep Neural Networks?" in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 27, 2014, pp. 3320–3328.

[17] F. Milletari, N. Navab, and S.-A. Ahmadi, "V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation," in *Proceedings of the Fourth International Conference on 3D Vision (3DV)*, 2016, pp. 565–571.

[18] I. Loshchilov and F. Hutter, "Decoupled Weight Decay Regularization," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

[19] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

[20] O. Russakovsky et al., "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.