Semantic Segmentation for Flood Recognition on UAV Visuals

Avi Vyas[1*], Amit Chauhan[1*†] and Rishit Chowdary[1*†]

[1]Department of Computer Science Engineering, BML Munjal University, 67th Milestone, NH 48, Kapriwas, 122413, Haryana, India.

*Corresponding author(s). E-mail(s): avi.vyas.21cse@bmu.edu.in; amit.chauhan.21cse@bmu.edu.in; rishitchowdary.gandi.21cse@bmu.edu.in;
†These authors contributed equally to this work.

Abstract

Our project aims to simplify post-disaster management while providing valuable insights into vulnerable areas. The project utilizes semantic segmentation techniques to analyze high-quality, low-resolution images captured by unmanned aerial vehicles (UAVs) in the aftermath of a disaster. We segment the images into two output classes that distinguish between flooded and non-flooded. Using the Custom modified dataset for our task, we train models of increasing complexity for segmentation and try to cover up the lackings of previous models. We used the good, old, reliable DeepLabV3 with ResNet backbone for semantic segmentation.

Keywords: DeepLabV3, Segmentation, ResNet, Flood

1 Introduction

Deep learning is emerging as an important tool for disaster risk management and post-disaster response. This push is also reflected in the goals of global organizations like the United Nations and World Bank, aiming to improve the infrastructure resiliency against shocks in developing countries by highlighting at-risk areas[1].An integral part of this analysis is identifying the worst affected areas from natural disasters and developing a decision support system. Computer vision and visual scene understanding provide a way to quickly interpret post-disaster scenarios to help design immediate

and effective relief while forming the basis for recognizing the weak points in the infrastructure.

Visual understanding datasets for post-disaster assess- ment generally come in three flavours :

- Satellite images [2]
- social media images [3]
- Unmanned aerial vehicle (UAV) images [4]

Satellite images are expensive to procure and suffer from low resolution and noise due to the high altitude. Social media images are low quality, noisy and not scalable for deep learning approaches. In light of these challenges, we decided to work with the third category of images to get high-resolution images taken from low altitudes.

In Aerial Images Semantic segmentation divides an image into homogeneous regions, which helps understand the image's local structure and identify boundaries (lines, curves, etc). More precisely, image segmentation is the process of assigning a label to every pixel in an image such that pixels with the same label share certain visual characteristics.

The inputs in our algorithms are images. We use DeepLabV3, convolution-based model and then we compare two other options for DeepLabV3 to predict the segmentation masks for the images :

- DeepLabV3 with ResNet50 backbone [5]
- DeepLabV3 with ResNet101 backbone [6]

The segmentation masks are the predictions given by the model specifying the class of each pixel in the image.

2 Related Work

For our project, we first survey the use of a simple UNet [7] architecture trained from scratch for the segmentation problem. UNets capture con- textual information through an autoencoder-decoder type architecture with skip-like connections between the down- sampling and upsampling layers. UNets show promising re- sults for segmentation tasks which rely on the global context for per-pixel segmentation with moderately large datasets. Since UNets do not necessarily capture local context level information, particularly for classes that are difficult to predict (like flooded vs non-flooded and smaller classes like the ve- hicle).

Our next approach was to try a Fully Convolutional Net- work (FCN) [8]. FCNs are an important class of models for semantic segmentation that fundamentally changed how people approached the problem, in addition to opening vari- ous avenues for future research. FCN replaces the fully connected layers at the end of an image classification network with upsampling followed by convolutional layers. This allows the features

learned from the image to be used for pixel level classification, rather than classifying the image as a whole.

DeepLabv3 [9][10] is one of the most popular state-of- the-art models used for semantic segmentation. It avoids losing spatial context caused by repeated pooling and striding using atrous convolutions. This is important for our dataset since the labelling of flood is based on the nearby context, specifically, whether it is surrounded by flood water. Further, deeplabv3 generates multi-scale feature maps which lead to greater performance when the objects in the image have different sizes, as is the case for our dataset. Hence, we use DeepLabv3 as our proposed models for this dataset.

Another Study[11] apply semantic segmentation on a dataset, named Volan2019, which includes 16 videos collected from recent natural disasters, containing people, flooded area, build- ing roof (damaged and undamaged), car, debris, vegetation, road, and boat. The dataset contains classes such as vehicles, roads, flooded areas and trees. This work experiments with Mask-RCNN and PSP- Net for their dataset.

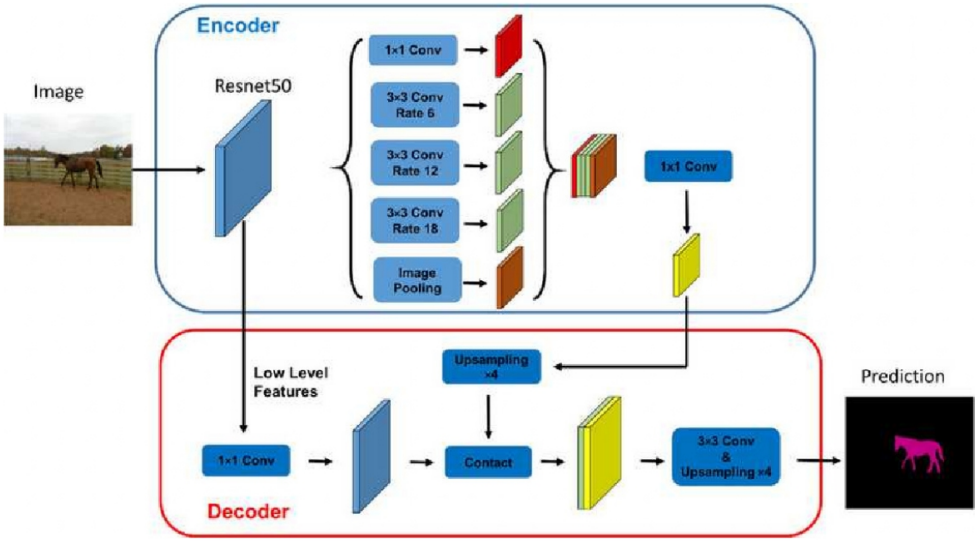

Fig. 1 Figure shows the network structure combining Deeplabv3 and ResNet 50.

Rahnemoonf studies[12] a related problem of detecting the flooded areas in Houston from data collected by UAVs. This work proposes an integration of densely connected RNN and CNN which addresses the difficulty in labelling smaller ob-jects and semantic ambiguities across object boundaries.Tt includes classes which have much smaller sizes than others (e.g. vehicles, ponds). The classification as flooded depends on whether it touches flood water on any side and hence improving the accuracy along object boundaries should be useful in improving performance.

In this paper[13], ResNet50 was used as the backbone for segmentation tasks, and the deep lab v3+ neural network structure composed of Atrous Spatial Pyramid Pooling (ASPP) and aux-classifier is added. The network input data input (H, W) are the same as the network output (H, W), significantly reducing the insufficiency

of data information caused by the limitation regarding the number of images. The model's performance on semantic segmentation tasks in the whole network is further improved by down-sampling and up-sampling. Figure 1 shows the basic structure of the neural network combining Deeplabv3 and ResNet 50.

ResNet50 has two basic blocks, named Conv-Block and Identity Block, respectively. The input and output dimensions of the Conv-Block are different, so they cannot be connected in series. Its function is to change the dimensions of the network. For the Identity Block, the dimensions of the input are the same as the dimensions of the output, and they can be connected in series to deepen the network.

3 Methodology

3.1 Data-Set

In this work, we use a modified version of the Flood Area Segmentation from Kaggle [14]. It's a pretty good and clean dataset. But it has some image extension issues and some of the images are corrupted.

We have prepared a modified version of this dataset, the dataset now is structured into Train and Validation Set below is the respresentation of new structure:

```
flood-area-segmentation
|-- train_images
|-- train_masks
|-- valid_images
`-- valid_masks
```

The Changes made to the dataset are:

- The dataset now contains a training and validation split.
- There are 257 training images & masks and 32 validation images & masks.
- There are no corrupted images in the dataset now.

All the ground truth images are in RGB format. And all the masks are grayscale images having white pixels (255) for the flood areas and black pixels (0) otherwise. So, it is a binary semantic segmentation dataset.

3.2 Pre-Processing

While we will not go through the entire methodology for dataset preparation, but there are a few important things to take notice of here.

One of them is the data augmentation strategy [15]

Remember that we have only 257 images and masks for training. This is not enough to train a good model. But data augmentation help us with this. We use the Albumentations library to easily apply the augmentations to the images and masks.

We apply the following augmentations to the training images and masks for the flood segmentation dataset :

- HorizontalFlip
- RandomBrightnessContrast
- RandomFog
- ImageCompression

Apart from horizontal flipping (applied with 0.5 probability), we apply all other augmentations with 0.2 probability.

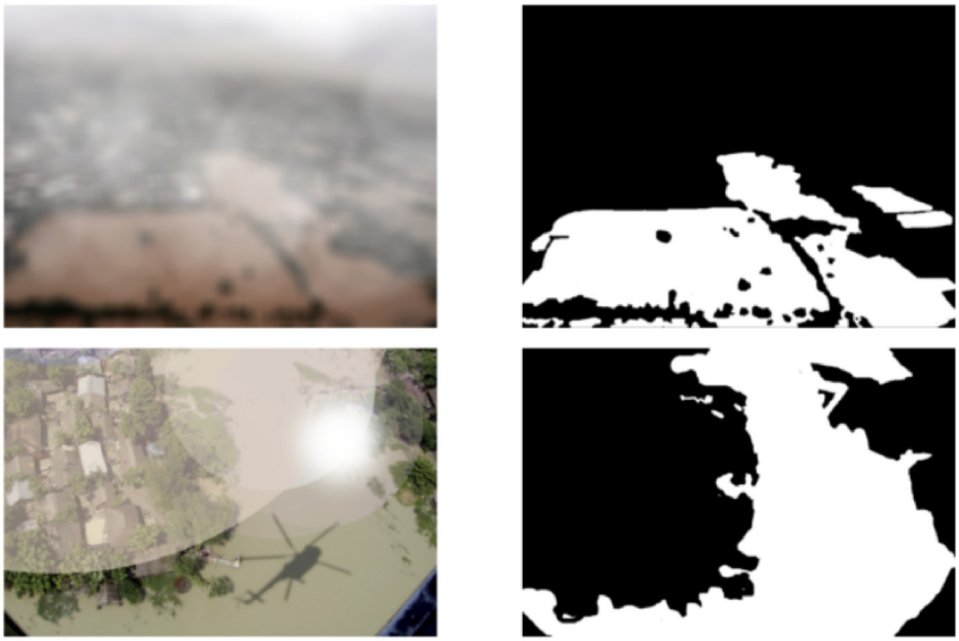

Fig. 2 This is what the images will look like after applying these augmentations randomly.

3.3 Model Architecture

We propose a two different architecture: DeepLabV3 with ResNet50 backbone and DeepLabV3 with ResNet101 backbone

```
DeepLabV3 with ResNet50 backbone Model Summary:
==========================================================================================================
Layer (type)                      Output Shape          Param #     Connected to
==========================================================================================================
InputLayer                        (None, 3, H, W)       -           -

IntermediateLayerGetter
  Conv2d                           (None, 64, H/2, W/2) 9472        InputLayer
  BatchNorm2d                      (None, 64, H/2, W/2) 128         Conv2d
  ReLU                             (None, 64, H/2, W/2) -           BatchNorm2d
  MaxPool2d                        (None, 64, H/4, W/4) -           ReLU

Sequential (layer1)
  Bottleneck
    Conv2d                          (None, 64, H/4, W/4) 4160        MaxPool2d
    BatchNorm2d                     (None, 64, H/4, W/4) 128         Conv2d
    ReLU                            (None, 64, H/4, W/4) -           BatchNorm2d
    Sequential (downsample)
      Conv2d                        (None, 256, H/4, W/4) 16640       Bottleneck Input
      BatchNorm2d                   (None, 256, H/4, W/4) 512         Conv2d

Sequential (layer2)
  Bottleneck
    Conv2d                          (None, 128, H/8, W/8) 32896       Sequential
    BatchNorm2d                     (None, 128, H/8, W/8) 256         Conv2d
    ReLU                            (None, 128, H/8, W/8) -           BatchNorm2d
    Sequential (downsample)
      Conv2d                        (None, 512, H/8, W/8) 66048       Bottleneck Input
      BatchNorm2d                   (None, 512, H/8, W/8) 1024        Conv2d

Sequential (layer3)
  Bottleneck
    Conv2d                          (None, 256, H/16, W/16) 131328    Sequential
    BatchNorm2d                     (None, 256, H/16, W/16) 512      Conv2d
    ReLU                            (None, 256, H/16, W/16) -        BatchNorm2d
    Sequential (downsample)
      Conv2d                        (None, 1024, H/16, W/16) 263168   Bottleneck Input
      BatchNorm2d                   (None, 1024, H/16, W/16) 2048    Conv2d

Sequential (layer4)
  Bottleneck
    Conv2d                          (None, 512, H/32, W/32) 65664     Sequential
    BatchNorm2d                     (None, 512, H/32, W/32) 1024     Conv2d
    ReLU                            (None, 512, H/32, W/32) -       BatchNorm2d
    Sequential (downsample)
      Conv2d                        (None, 2048, H/32, W/32) 1050624 Bottleneck Input
      BatchNorm2d                   (None, 2048, H/32, W/32) 2048    Conv2d

DeepLabHead (classifier)
  ASPP
    Conv2d                          (None, 256, H/16, W/16) 524544   layer4
    BatchNorm2d                     (None, 256, H/16, W/16) 512      Conv2d
    ReLU                            (None, 256, H/16, W/16) -       BatchNorm2d
    Conv2d                          (None, 2, H/16, W/16) 514         ASPP Output

FCNHead (aux_classifier)
  Conv2d                            (None, 256, H/16, W/16) 262400   layer3
  BatchNorm2d                       (None, 256, H/16, W/16) 512      Conv2d
  ReLU                              (None, 256, H/16, W/16) -       BatchNorm2d
  Conv2d                            (None, 2, H/16, W/16) 514         Conv2d

==========================================================================================================
Total params: 41,994,308
Trainable params: 41,994,308
Non-trainable params: 0
```

DeepLabV3 ResNet50 Model Architecture Overview:

- **Backbone (ResNet50)**: The model begins with a ResNet50 backbone, which consists of deep residual blocks. ResNet architectures are known for mitigating the vanishing gradient problem in deep networks, allowing for the training of very deep neural networks.
- **DeepLabV3 Head**: The DeepLabV3 head is appended to the ResNet50 backbone. This head consists of atrous convolutional layers to capture multi-scale contextual information. Atrous convolutions, also known as dilated convolutions, enable the network to have a broader field of view without increasing the number of parameters.

- **Classifier Modification**: The classifier layer (classifier[4]) is modified to output the desired number of classes. Specifically, the number of output channels is adjusted to match the specified num_classes. This layer is responsible for producing the final segmentation map.
- **Auxiliary Classifier Modification**: Similar to the main classifier, the auxiliary classifier layer (aux_classifier[4]) is also modified to output the specified number of classes. This auxiliary classifier is often employed during training to provide additional supervision signals.

```
DeepLabV3 with ResNet101 backbone Model Summary:
==========================================================================================
Layer (type)                      Output Shape              Param #     Connected to
==========================================================================================
InputLayer                        (None, 3, H, W)           0           -
Conv2d-1                          (None, 64, H/2, W/2)      9472        InputLayer
BatchNorm2d-1                     (None, 64, H/2, W/2)      128         Conv2d-1
ReLU-1                            (None, 64, H/2, W/2)      0           BatchNorm2d-1
MaxPool2d-1                       (None, 64, H/4, W/4)      0           ReLU-1
Bottleneck-1                      (None, 256, H/4, W/4)     -           MaxPool2d-1
  Conv2d-2                        (None, 64, H/4, W/4)      4096        Bottleneck-1 Input
  BatchNorm2d-2                   (None, 64, H/4, W/4)      128         Conv2d-2
  Conv2d-3                        (None, 64, H/4, W/4)      36864       BatchNorm2d-2
  BatchNorm2d-3                   (None, 64, H/4, W/4)      128         Conv2d-3
  Conv2d-4                        (None, 256, H/4, W/4)     16384       BatchNorm2d-3
  BatchNorm2d-4                   (None, 256, H/4, W/4)     512         Conv2d-4
  ReLU-2                          (None, 256, H/4, W/4)     0           BatchNorm2d-4
  Sequential-1                    (None, 256, H/4, W/4)     0           ReLU-2
  ...
  ...p
Bottleneck-16                     (None, 2048, H/32, W/32)  -           -
  Conv2d-50                       (None, 512, H/32, W/32)   1048576     Bottleneck-16 Input
  BatchNorm2d-50                  (None, 512, H/32, W/32)   1024        Conv2d-50
  Conv2d-51                       (None, 512, H/32, W/32)   2359296     BatchNorm2d-50
  BatchNorm2d-51                  (None, 512, H/32, W/32)   1024        Conv2d-51
  Conv2d-52                       (None, 2048, H/32, W/32)  1048576     BatchNorm2d-51
  BatchNorm2d-52                  (None, 2048, H/32, W/32)  4096        Conv2d-52
  ReLU-16                         (None, 2048, H/32, W/32)  0           BatchNorm2d-52
  Sequential-16                   (None, 2048, H/32, W/32)  0           ReLU-16
  ...
  ...
ASPP                              (None, 256, H/32, W/32)   -           Bottleneck-16
  Conv2d-53                       (None, 256, H/32, W/32)   524288      ASPP
  BatchNorm2d-53                  (None, 256, H/32, W/32)   512         Conv2d-53
  ReLU-17                         (None, 256, H/32, W/32)   0           BatchNorm2d-53
  ...
  ...
Conv2d-54                         (None, 2, H/32, W/32)      514         ASPP
==========================================================================================
Total params: 60,986,436
Trainable params: 60,986,436
Non-trainable params: 0
```

DeepLabV3 ResNet101 Model Architecture Overview:

- **Backbone (ResNet101)**: The model starts with a ResNet101 backbone, which is deeper than ResNet50. This increased depth allows the network to capture more intricate features and representations from the input images.
- **DeepLabV3 Head**: Similar to the ResNet50 variant, the DeepLabV3 head is added after the ResNet101 backbone. This head contains atrous convolutions to maintain a large receptive field.
- **Classifier Modification**: The classifier layer is modified, as in the ResNet50 variant, to output the specified number of classes.

- **Auxiliary Classifier Modification**: Like the ResNet50 variant, the auxiliary classifier layer is modified to output the desired number of classes for additional supervision during training.

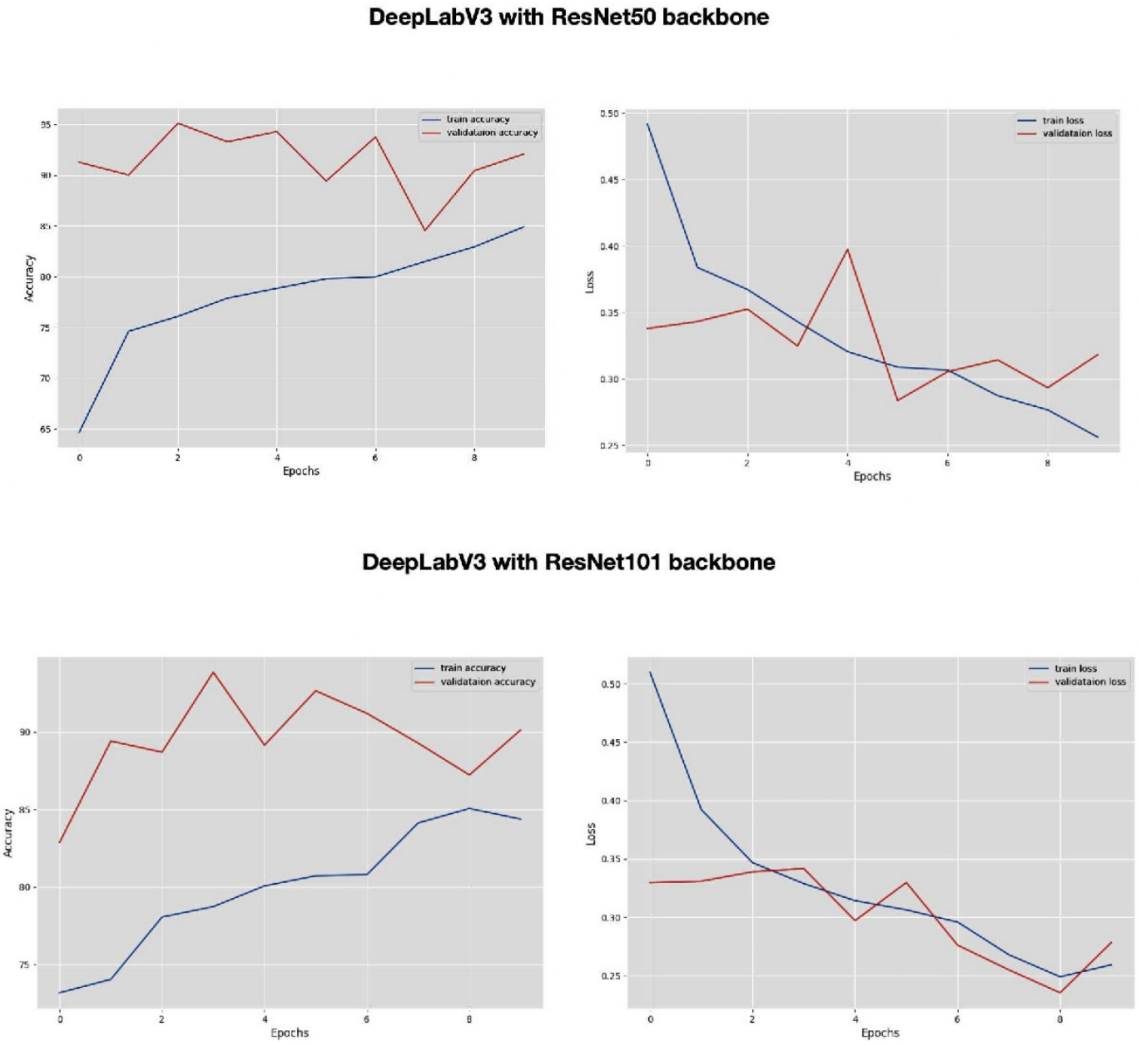

Fig. 3 Figure showing the Accuracy and loss of DeepLabV3 with ResNet50 backbone and DeepLabV3 with ResNet101 backbone respectively

Note: All training and inference experiments were carried on a Colab free with T4.

Hyperparameters:

- **epochs**: The number of epochs that we want to train the model.
- **lr**: The learning rate for the optimizer. It is 0.0001 by default.
- **batch**: Batch size for the data loader.
- **imgsz**: The image size we want to train the model with. Semantic segmentation models often need higher resolution images to achieve good results. We will train the model on 512×512 images which is the default value as well.
- **scheduler**: This is a boolean flag indicating whether we want to use the learning rate scheduler or not. If we pass this flag then the learning rate will be reduced by a factor of 10 after 25 epochs.

If you face OOM (Out Of Memory) error while training, please consider reducing the image size or the batch size.

Here,we were training the model for 50 epochs, with a batch size of 4, starting learning rate of 0.0001, and also applying the learning rate scheduler.

Here are the truncated outputs from the terminal.

```
python train.py --epochs 50 --batch 4 --lr 0.0001 --scheduler
Namespace(epochs=50, lr=0.0001, batch=4, scheduler=True)
41,994,308 total parameters.
41,994,308 training parameters.
Adjusting learning rate of group 0 to 1.0000e-04.
EPOCH: 1
Training
Loss: 0.4229 | PixAcc: 83.23: 100%|                                                    | 64/64
      ↪ [00:50<00:00,  1.26it/s]
Validating
Loss: 0.3060 | PixAcc: 89.33: 100%|                                                    | 8/8
      ↪ [00:02<00:00,  3.57it/s]
Best validation loss: 0.38311174884438515
Saving best model for epoch: 1
Train Epoch Loss: 0.5018, Train Epoch PixAcc: 64.9515
Valid Epoch Loss: 0.3831, Valid Epoch PixAcc: 86.8635
Adjusting learning rate of group 0 to 1.0000e-04.
------------------------------------------------
EPOCH: 2
Training
Loss: 0.3776 | PixAcc: 81.60: 100%|                                                    | 64/64
      ↪ [00:49<00:00,  1.30it/s]
Validating
Loss: 0.4301 | PixAcc: 91.83: 100%|                                                    | 8/8
      ↪ [00:02<00:00,  3.84it/s]
Train Epoch Loss: 0.3887, Train Epoch PixAcc: 72.3201
Valid Epoch Loss: 0.4537, Valid Epoch PixAcc: 94.0050
Adjusting learning rate of group 0 to 1.0000e-04.
------------------------------------------------
EPOCH: 3
Training
Loss: 0.4062 | PixAcc: 68.83: 100%|                                                    | 64/64
      ↪ [00:49<00:00,  1.29it/s]
Validating
Loss: 0.2974 | PixAcc: 93.10: 100%|                                                    | 8/8
      ↪ [00:02<00:00,  3.86it/s]
Train Epoch Loss: 0.3698, Train Epoch PixAcc: 75.0250
Valid Epoch Loss: 0.4060, Valid Epoch PixAcc: 94.7763
Adjusting learning rate of group 0 to 1.0000e-04.
------------------------------------------------
.
.
.
EPOCH: 50
Training
Loss: 0.1831 | PixAcc: 80.59: 100%|                                                    | 128/128
      ↪ [00:39<00:00,  3.26it/s]
Validating
Loss: 0.1339 | PixAcc: 98.66: 100%|                                                    | 16/16
      ↪ [00:02<00:00,  7.16it/s]
Train Epoch Loss: 0.1459, Train Epoch PixAcc: 91.4595
Valid Epoch Loss: 0.2437, Valid Epoch PixAcc: 88.4480
Adjusting learning rate of group 0 to 1.0000e-06.
------------------------------------------------
TRAINING COMPLETE
```

4 Discussion

Let's talk about why flood segmentation is difficult in terms of this dataset.

First of all, this dataset contains only 257 training images and masks. Surely, this is not enough to train a state-of-the-art semantic segmentation model.Still we implemented ample augmentation techniques just to keep the loss reducing.

Secondly, there were many small background pixels in between the flood areas. These can range from houses to agricultural fields, and even partially submerged vehicles.

The model have to learn these background classes properly and not segment them. For smaller models, this is particularly challenging.

5 Results

The performance of the segmentation in images is tested through the following metric to compare the predicted segmentation masks and the ground truth segmentation masks present in the dataset and evaluate the performance of the both the models.

- **Pixel Accuracy** is calculated by determining the fraction of pixels that are correctly classified by the model out of the total number of pixels in the image. This metric provides a straightforward assessment of the model's ability to precisely identify and assign the correct segmentation labels to individual pixels.

$$P = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

i.e.

$$P = \frac{N}{M} \tag{2}$$

where,

N = *Number of Correctly Classified Pixels*
M = *Total Number of Pixels In The Image*

Table 1 Pixel accuracy on validation and train sets with data augmentation.

	Model A [1]		Model B [2]	
Epochs	Train Acc	Val Acc	Train Acc	Val Acc
10	79.8005	89.4173	85.0673	87.2264
50	82.4122	86.4420	87.8898	85.4121

Here are some of the outputs. The first column shows the predictions by the model, and the right column shows the ground truth masks.

[0]Note : The results displayed represent the best models saved during each epoch iteration, capturing the optimal state when the loss was at its lowest. These do not necessarily reflect the complete final model after the specified number of epochs.
[1]DeepLabV3 with ResNet50 Backbone
[2]DeepLabV3 with ResNet101 Backbone

Table 2 Loss on validation and train sets with data augmentation.

	Model A [1]		Model B [2]	
Epochs	Train Loss	Val Loss	Train Loss	Val Loss
10	0.3087	0.2836	0.2490	0.2353
50	0.2682	0.2223	0.2401	0.2312

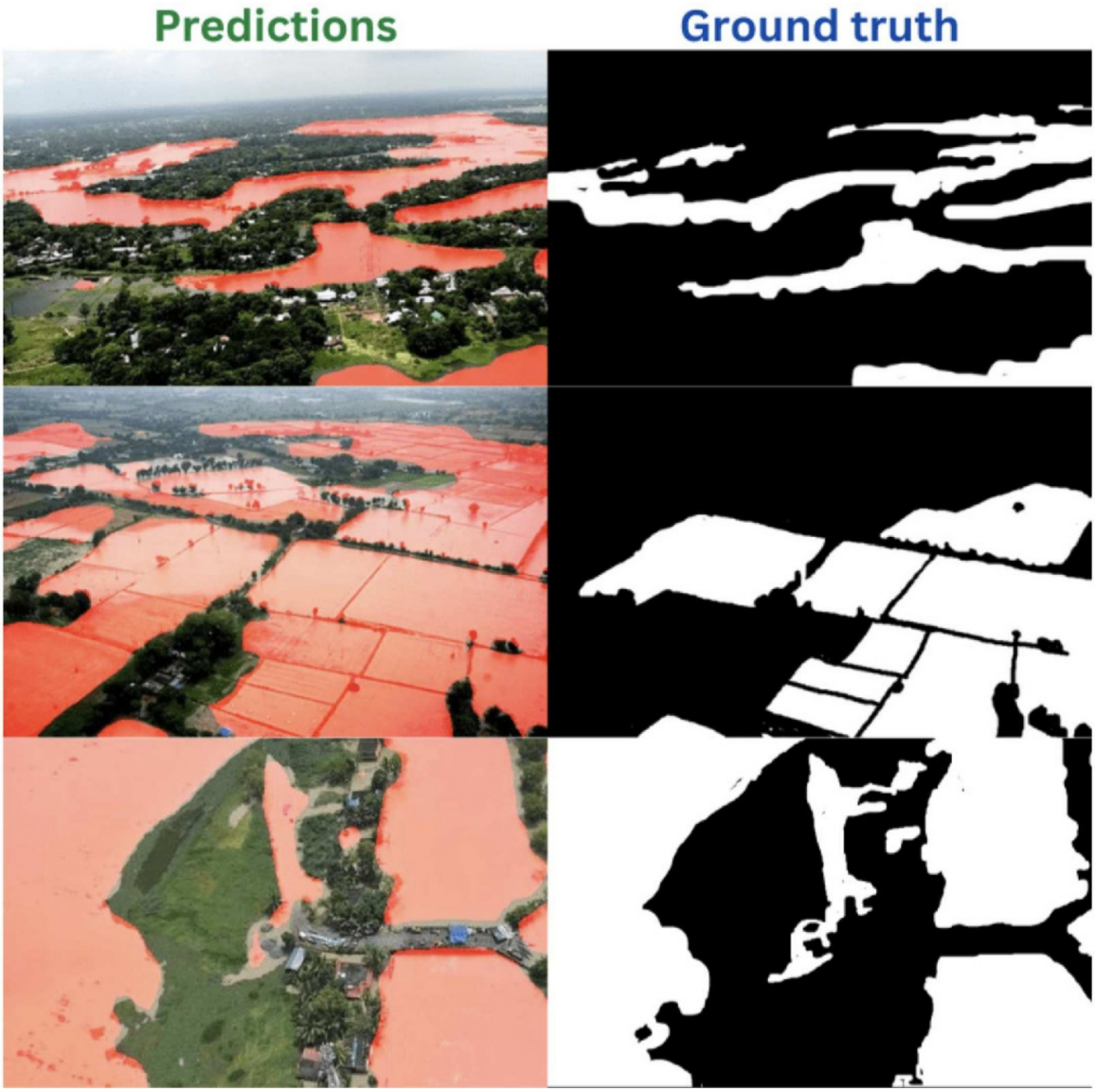

Fig. 4 Predictions for flood segmentation

6 Conclusion

In this work, we aimed to segment flood water in UAV visual using our modified dataset, We experimented and trained two model namely DeepLabV3 with ResNet50

[0]Note : The results displayed represent the best models saved during each epoch iteration, capturing the optimal state when the loss was at its lowest. These do not necessarily reflect the complete final model after the specified number of epochs.
[1]DeepLabV3 with ResNet50 Backbone
[2]DeepLabV3 with ResNet101 Backbone

and DeepLabV3 with ResNet101 Backbone and compared them owing to their performance on common semantic segmentation benchmark. During experimentations, we identified that the models still finds it difficult to differentiate between thin lines of land and the flooded area. But that is something that we can rectify with more training data and better training techniques.

Based on our results, we would prefer DeepLabV3 with ResNet50 overall and for Video based application and DeepLabV3 with ResNet100 for Images only but not for video.

The main reason for choosing DeepLabV3 with ResNet50 over ResNet100 is the speed and performance. With ResNet101 backbone, While it performs well, it requires more memory to train and is also slower during inference which is itself an issue when working with videos.

Our initial results are promising,Surely, the model is still segmenting very small patches of land as water. But the results are really good considering we had only 257 training images and we see a general trend of improvement in our metrics for the validation and train set. Running our models for more epochs and continuing with a learning rate schedule will improve our results for this problem.

7 Future Scope

- Implementation of Semi-Supervised Learning reason being there is only small portion of labeled data and lots of unlabeled data for this type of application.
- This project has a lot of potential if we can scale the training data and the training pipeline. Such a deep learning model can be attached to a drone to segment out the flooded areas in a region. This will help make proper plans to restructure a particular region so that such flood situations do not occur in the future.

Declarations

- **Conflict of interest** The authors declare that they have no conflict of interest.

Appendix A

Our code is available on Github at https://github.com/avi7410/FLOOD_SEGMENTATION_UAV

References

[1] Frontline: Preparing healthcare systems for shocks from disasters to pandemics. https://openknowledge.worldbank.org/handle/10986/35429

[2] Bischke, B., Helber, P., Schulze, C., Srini- vasan, V., Dengel, A., Borth, D.: The multimedia satellite task at mediaeval 2017. In: MediaEval Benchmark (2017)

[3] Weber, E., Marzo, N., Papadopoulos, D.P., Biswas, A., Lapedriza, A., Ofli, F., Imran, M., Torralba, A.: Detecting natural disasters, damage, and incidents in the wild. In: Computer Vision – ECCV 2020: 16th European Conference, pp. 331–350. Springer, ??? (2020)

[4] Rahnemoonfar, M., Chowdhury, T., Sarkar, A., Varshney, D., Yari, M., Murphy, R.R.: Floodnet: A high resolution aerial imagery dataset for post flood scene understanding. IEEE Access **9**, 89644–89654 (2021)

[5] https://pytorch.org/vision/main/models/generated/torchvision.models.segmentation.deeplabv3_resnet50.html

[6] https://pytorch.org/vision/main/models/generated/torchvision.models.segmentation.deeplabv3_resnet101.html

[7] Ronneberger, O., Fischer, P., Brox, T.: U-net: Convolutional networks for biomedical image segmentation. In: Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015, pp. 234–241 (2015)

[8] Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2015)

[9] Chen, L.-C., Papandreou, G., Schroff, F., Adam, H.: Rethinking atrous convolution for semantic image segmentation. arXiv preprint (2017) 1706.05587

[10] Chen, L.-C., Zhu, Y., Papandreou, G., Schroff, F., Adam, H.: Encoder-decoder with atrous separable convolution for semantic image segmentation. In: ECCV (2018)

[11] Pi, Y., Nath, N.D., Behzadan, A.H.: Detection and semantic segmentation of disaster damage in uav footage. Journal of Computing in Civil Engineering **35**(2), 04020063 (2021)

[12] Rahnemoonfar, M., Murphy, R., Vicens Miquel, M., Dobbs, D., Adams, A.: Flooded area detection from uav images based on densely connected recurrent neural networks. In: IGARSS 2018 - 2018 IEEE International Geoscience and Remote Sensing Symposium, pp. 1788–1791 (2018)

[13] Chen, Y.K..P.G..S.F..A.H. L.C.; Zhu: Encoder-decoder with atrous separable

convolution for semantic image segmentation. (2018) https://doi.org/10.48550/arXiv.1802.02611.

[14] https://www.kaggle.com/datasets/faizalkarim/flood-area-segmentation

[15] Yang, S., Xiao, W., Zhang, M., Guo, S., Zhao, J., Shen, F.: Image data augmentation for deep learning: A survey https://doi.org/10.48550/arXiv.2204.08610

DIP

ORIGINALITY REPORT

4%	2%	2%	0%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	"Computer Vision – ECCV 2018", Springer Science and Business Media LLC, 2018 Publication	2%
2	ia-petabox.archive.org Internet Source	2%

Exclude quotes	On	Exclude matches	< 6 words
Exclude bibliography	On		