geo.ai: Object Extraction from Satellite Images

U-Net convnet with LCNN loss

1st Siddhant Sharma
Department of Electrical Engineering
Indian Institute of Technology Delhi
New Delhi, India
Siddhant.Sharma.ee119@ee.iitd.ac.in

2nd Mohd Zaki
Department of Civil Engineering
Indian Institute of Technology Delhi
New Delhi, India
mohd.zaki@civil.iitd.ac.in

3rd Sandeep Kumar
Department of Electrical Engineering
Indian Institute of Technology Delhi
New Delhi, India
ksandeep@iitd.ac.in

4th N. M. Anoop Krishnan
Department of Civil Engineering
Indian Institute of Technology Delhi
New Delhi, India
krishnan@iitd.ac.in

5th Jayadeva
Department of Electrical Engineering
Indian Institute of Technology Delhi
New Delhi, India
jayadeva@iitd.ac.in

***Abstract*—The researchers have been remotely sensing the earth using hot air balloons, reconnaissance airplanes, and satellites. Due to lack of computational facilities, it has always been a challenging task to accurately demarcate boundaries of important structures like water bodies, agricultural lands and forests to name a few. In this work, we present deep learning based architecture U-Net and apply transfer learning for semantic segmentation of satellite images. Specifically, we have fine-tuned the existing model on satellite imagery to identify electrical substations. For the first time, LCNN loss has been included to improve the predictions of U-Net.**

***Index Terms*—Remote sensing, satellite imagery, semantic segmentation, deep learning**

I. Introduction

Technological advancements in the field of electronics, satellites, computing and artificial intelligence opened up a vast area for application of machine learning in the field of satellite imagery. Also, the availability of open source satellite images provided by Bhuvan [1], has enabled researchers to study the forests [2], irrigation source and water bodies [3], and monitor fires in agricultural sites with the help of Internet of Things and deep learning [4].

In this competition, the participants were give the taks of segmenting the electrical substations from the satellite images. Here, we demonstrate the capabilities of deep learning and transfer learning using a pre-trained deep convolutional architecture called U-Net and achieve an accuracy of 0.929 on training data, 0.959 on validation augmented data and 0.755 on the test data.

A. Equations

The loss function used in the model is Dice Loss alongside LCNN (and m-LCNN; modified LCNN), the formula for Dice Loss is:

$$DiceLoss(A,B) = 1 - \frac{2\cdot|A\cap B|}{|A|+|B|} \tag{1}$$

And the LCNN [5] loss is:

$$LCNN = \sum_{i=1}^{M}\sum_{j=1}^{l_h}(w_{h_j}^T\cdot u_{h-1}^i + b_{h_j})^2 \tag{2}$$

Similarly the modified LCNN (m-LCNN) is:

$$m-LCNN = \sum_{i=1}^{M}\sqrt{\sum_{j=1}^{l_h}(w_{h_j}^T\cdot u_{h-1}^i + b_{h_j})^2} \tag{3}$$

And hence we can see that the total loss function is (for model 10, model 11 and model 12):

$$Loss(A,B) = 1 - \frac{2\cdot|A\cap B|}{|A|+|B|} + c\cdot\sum_{i=1}^{M}\sum_{j=1}^{l_h}(w_{h_j}^T\cdot u_{h-1}^i + b_{h_j})^2 \tag{4}$$

or it is (for model 8 and model 9):

$$Loss(A,B) = 1 - \frac{2\cdot|A\cap B|}{|A|+|B|} + \sum_{i=1}^{M}\sqrt{\sum_{j=1}^{l_h}(w_{h_j}^T\cdot u_{h-1}^i + b_{h_j})^2} \tag{5}$$

For other models, the loss function was just the Dice Loss.

And the sanity metric was the Jaccard score, also known as IoU (intersection over union):

$$IoU(A,B) = \frac{|A\cap B|}{|A\cup B|} \tag{6}$$

II. Methods Used

A. Data Set Pruning and Augmentation

1) Data Set: 100 $750\times750\times3$ images were given which contained satellite images of electrical substations and their masks stored as a polygon in a csv file.

The polygon masks were then changed to a binary mask of 750x750x1 size as the final outputs requested from the model was a binary mask as well.

Additionally as a pretrained U-Net model [6] was used with input size $256 \times 256 \times C$, where C are the number of channels, so the images were resized from $750 \times 750 \times 3$ to $256 \times 256 \times 3$ to be compatible with the pretrained model and also since a super sharp image was not required for this task and only increased computational time.

2) Data Augmentation: Due to the lack of images, augmenting the data was of utmost importance. In each of the 7 models trained, the method of augmentation was different to further diversify the inputs images we fed the model. But each model ($1 \rightarrow 5$ and $8 \rightarrow 11$) contained 400 additional augmented training samples on top of the 100 original training samples while model 6 and 7 contained just the 100 original training samples with no augmentation for training samples.

The various methods of augmentation are as follows:

- CoarseDropout / Dropout : blackens random patches / pixels of the image. This was applied to 200/1700 images.
- ElasticTransformations : makes the image get a water like murkiness. This was applied to 200/1700 images.
- Rotate / Flip / Scale / Translate : geometric transformations that change both the image and the mask. These were applied to 1200/1700 images and is the most effective form of augmentation.
- Changing hue / brightness / contrast / sharpen image : changes the values of the pixels of the image and hence do not change the mask. These were applied to 1000/1700 images to boost accuracy.

It is to be noted that just by doing the above augmentation, we were able to jump from a Jaccard Score of ~ 0.5 to $0.85 \uparrow$ and hence it is a very powerful technique.

Additionally, the cross validation set was also created solely from original training images by carefully augmenting the images with flips and crops.

3) Data Pruning: It was noticed that the model was performing poorly on 11 ambiguous images where the electric substations were hidden due to trees or surrounding houses. Just removing such cases increased the Jaccard Score by atleast 0.03 on average among the 7 models.

Those images were then singled out and a different technique of masking using shape files was used while training on them.

B. Model Selection and Training

1) Selecting model: We required an architecture to predict regions accurately. The top contenders for this task were Mask-RCNN [7], modified GNNs for region proposals [8], and U-Net [9] architecture. The reason for choosing U-Net that has generally been used for brain imaging tasks is due to the fact that the architecture of U-Net allows for concatenation of low level feature maps and higher ones. This allows for a more robust region to be predicted for the object detection task.

2) Pretrained U-Net Model: The official pretrained U-Net model of pytorch [6] was used for this competiton. It was seen that the pretrained model was significantly better than training from scratch for predicting electrical substations as well. The official model started from 32 channels and went

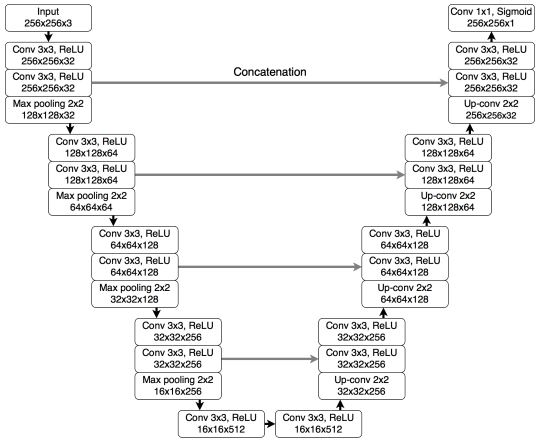


Fig. 1. U-Net architecture [6]

to 512 channels in the bottleneck layer. It input an image of $256 \times 256 \times C$ dimensions and output a 256×256 mask. Each block of the U-Net contained two convolutional layers followed by batch normalization with an ReLU activation function. The block ended with either a max pooling layer or a up-convolutional layer if it were the encoding part or the decoding respectively. And as is common in all U-Net architectures, skip connections from the encoding part were made to the decoding part of the architecture.

3) Model training: Model was trained by optimising the Dice Loss (along with LCNN / m-LCNN for some models) while using Jaccard Score as a metric. The optimizer was Adam with learning rates that differed for each model. Validation was done on the augmented train data set itself due to the lack of input data given.

An ensemble of models were trained, with the below hyperparameters:

- Model 1 : Learning Rate = $1e-4$, Epochs = 100, Batch Size = 64, Training Size = 500, Cross Validation Size = 100, Seed = Random, Built upon pretrained model.
- Model 2 : Learning Rate = $2e-5$, Epochs = 100, Batch Size = 64, Training Size = 500, Cross Validation Size = 100, Seed = 10007, Built upon Model 1.
- Model 3 : Learning Rate = $2e-5$, Epochs = 100, Batch Size = 64, Training Size = 500, Cross Validation Size = 100, Seed = 1009, Built upon Model 1.
- Model 4 : Learning Rate = $4e-5$, Epochs = 50, Batch Size = 64, Training Size = 500, Cross Validation Size = 100, Seed = 10531, Built upon Model 2.
- Model 5 : Learning Rate = $4e-5$, Epochs = 50, Batch Size = 64, Training Size = 500, Cross Validation Size = 100, Seed = 10433, Built upon Model 3.
- Model 6 : Learning Rate = $2e-4$, Epochs = 50, Batch Size = 32, Training Size = 100, Cross Validation Size = 100, Seed = 10711, Built upon Model 4.
- Model 7 : Learning Rate = $2e-4$, Epochs = 50, Batch Size = 32, Training Size = 100, Cross Validation Size = 100. Seed = 10091, Built upon Model 5.
- Model 8 : Learning Rate = $2e-4$, Epochs = 20, Batch

Size = 64, Training Size = 500, Cross Validation Size = 100. Seed = 10069, Built upon Model 4 and uses m-LCNN loss + Dice Loss.

- Model 9 : Learning Rate = $2e-4$, Epochs = 20, Batch Size = 64, Training Size = 500, Cross Validation Size = 100. Seed = 10103, Built upon Model 5 and uses m-LCNN loss + Dice Loss.
- Model 10 : Learning Rate = $4e-4$, Epochs = 40, Batch Size = 64, Training Size = 500, Cross Validation Size = 100. Seed = 21011, Built upon Model 9 and uses LCNN loss + Dice Loss with $c = 0.1$ as weight for LCNN.
- Model 11 : Learning Rate = $5e-4$, Epochs = 40, Batch Size = 64, Training Size = 500, Cross Validation Size = 100. Seed = 21023, Built upon Model 8 and uses LCNN loss + Dice Loss with $c = 0.4$ as weight for LCNN.
- Model 12: Learning Rate = $2e-4$, Epochs = 40, Batch Size = 32, Training Size = 100, Cross Validation Size = 100. Seed = 10091, Built upon Model 2 and uses LCNN loss + Dice Loss with $c = 0.4$ as weight for LCNN. Models built after epoch 10 and epoch 25 were also saved (as Model 12.1 and 12.2) to use for ensemble.

All of the above models had the same Image Size of $256 \times 256 \times 3$ as defined by the pretrained U-Net and the first layer was frozen for all of them (and hence would not train).

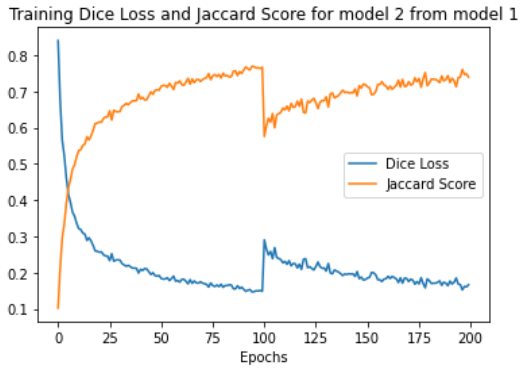
Additionally, each of the above Models had very different training sets due to the different types and extent of augmentation on them.

III. Results

Add graphs for all 7+3 models and the validation curves for all that are there. Add their (best) jaccard scores and dice loss as well on the OG training set if available All models (and ensembles) generated are important as either they perform better in some cases than others or they are the building blocks of the other models. Hence it is important to show and analyse all models.

A. Model 1

We can see that after 100 epochs, we need to change a few hyperparameters to hasten the convergence for the model.

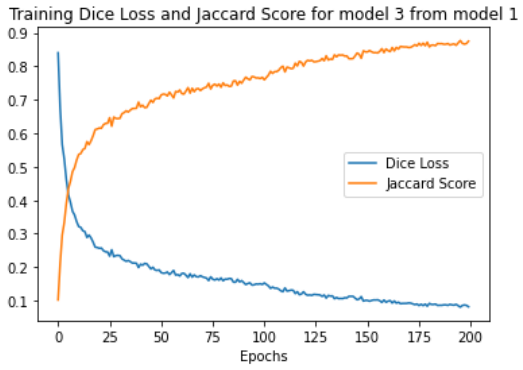


B. Model 2

Building upon model 1, we can see a sharp drop in accuracy and an increase in loss. This is due to the drastic augmentations done in model 2, to increase validation and test accuracy.

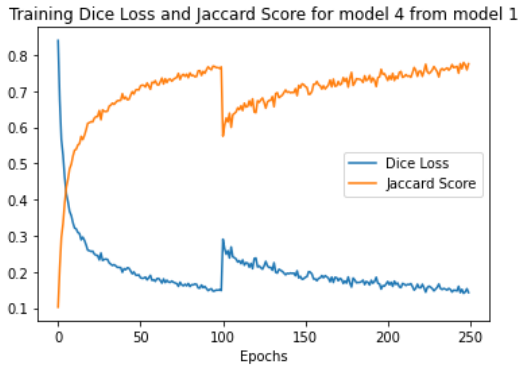
Despite having similar loss and accuracy as model 1, this model is considerably better on test cases.

C. Model 3

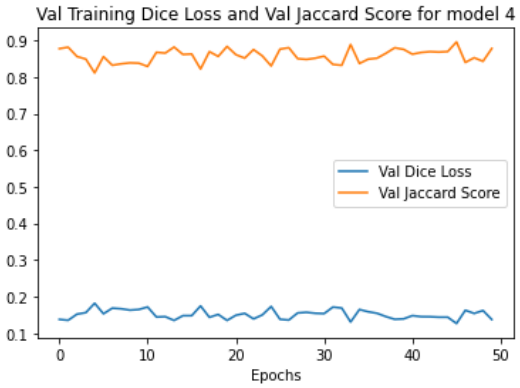


While building similarly from model 1, like model 2, this model had considerably less drastic augmentations and relied on more softer geometric augmentations. This is why the loss curve is smoother than for model 2.

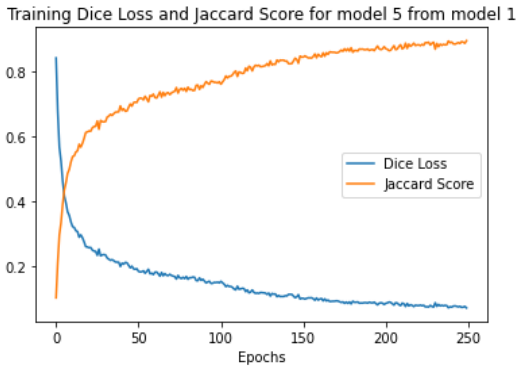
D. Model 4



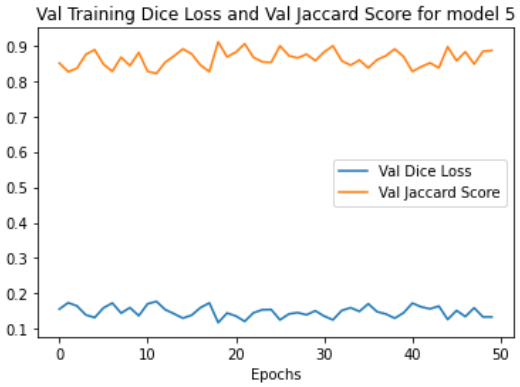
Model 4 builds on top of model 2, and hence has trained for 200 epochs on top of the pretrained model. The model is still achieving better scores as compared to model 2 and hence the fear of overfitting creeps in. But we can be assured that the model doesn't overfit by noticing that the validation loss is still decreasing ever so slightly.

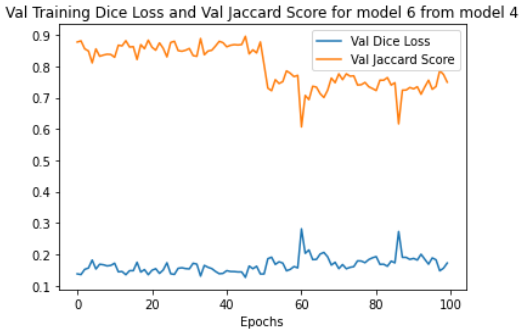


E. Model 5



Similar to model 4, model 5 builds on from model 3 and has been trained for 200 epochs on top of the original pretrained model. But like model 4, one look at validation loss curve is enough to ascertain that model 5 doesn't overfit.

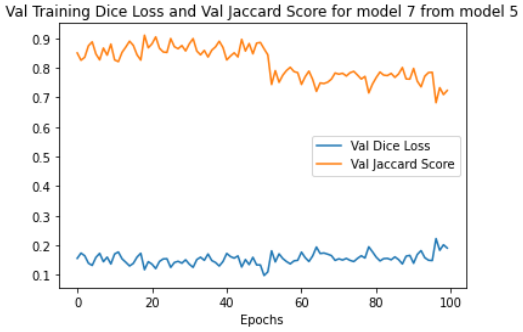




F. Model 6

Just by the validation loss curve, we can see that our model has started overfitting. Hence model 6 is worse than model 4, that it was build upon.

G. Model 7

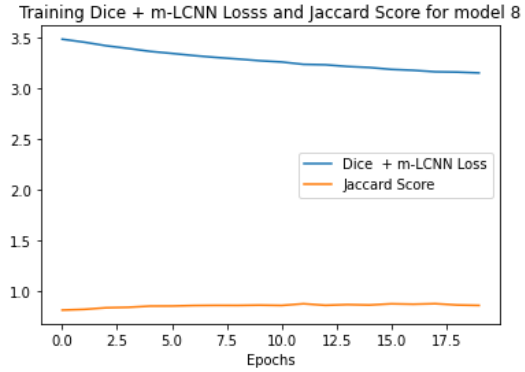


With a similar argument to model 6, model 7 has also overfitted and it makes sense to drop this model (from all future ensembles as well)

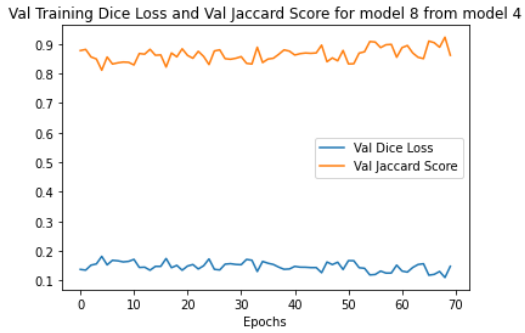
H. Model 8

Starting from Model 8, we started using more sophisticated loss functions but due to the sudden change in it's value Jaccard Score would be a better metric to plot.

Since we can see that the sanity metric, the Jaccard Score has still been increasing, our model is getting better. Valida-

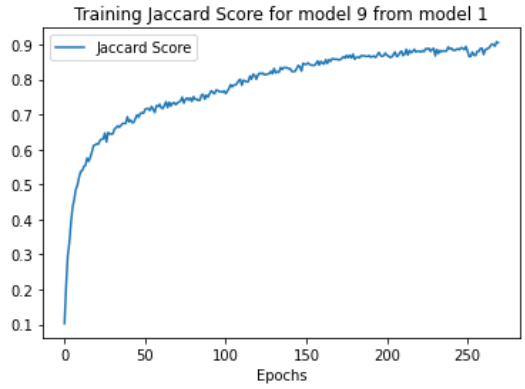


tion loss doesn't include m-LCNN loss for more clarity and by plotting it, we can see that our model is getting better on the validation set as well. Additionally, it is worth noting



that m-LCNN loss (and subsequent models with LCNN loss as well) is extremely useful in achieving even higher scores when previous methods had plateaued.

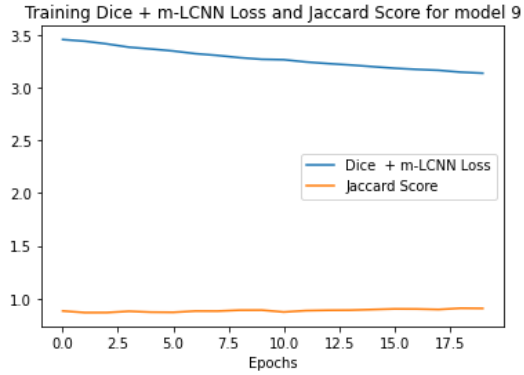
I. Model 9

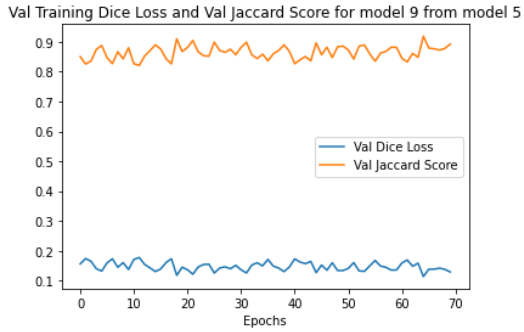


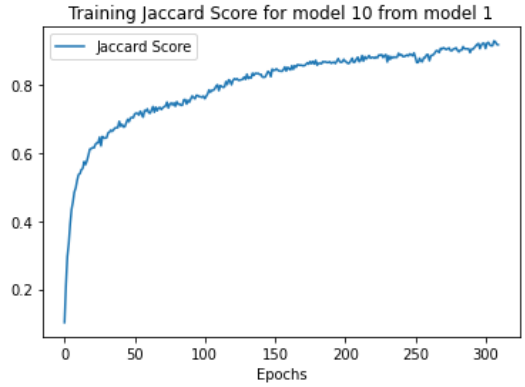
Similar to Model 8, we see further growth in the Jaccard Score just due to the addition of the m-LCNN term.

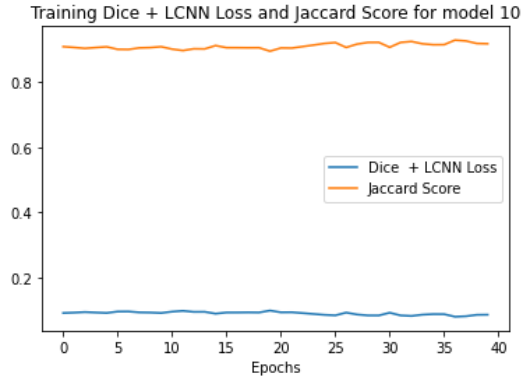
J. Model 10

Model 10 (and 11) included the LCNN loss, and we can see that despite being trained for 310 epochs the Jaccard Score is still increasing.



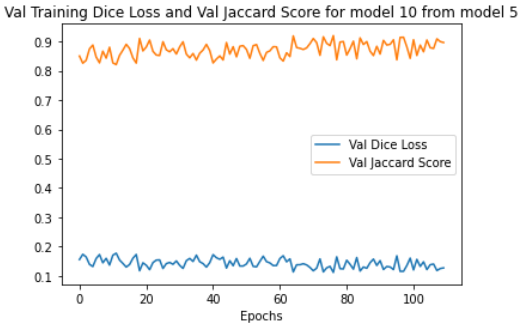




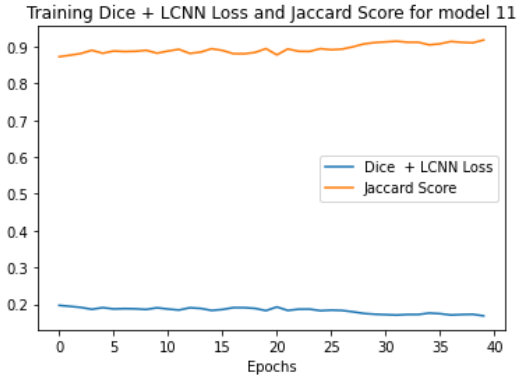


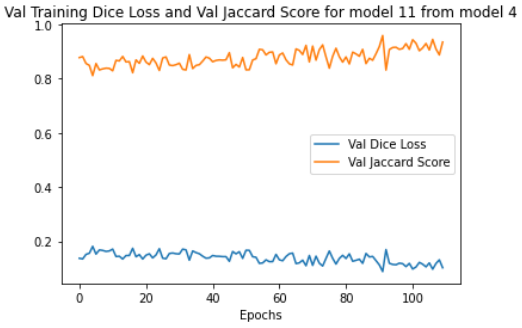
K. Model 11

Due to having a higher weight for the LCNN loss, the loss decrease was more apparent in Model 11 when compared to



the same in Model 10.





L. Ensemble Models

To further better the predictions, the best models (namely model 2,4,5,8,9,10,11) were taken in an ensemble using the below methods.

1) Sum Ensemble Model: All predictions were simply added up and then rounded off to $\{0, 1\}$. This caused a lot of false positives but also had the highest true positives amongst all ensemble models.

2) Average Ensemble Model: The average ensemble is just the sum ensemble but all values were divided by the number of models before rounding them off. Due to the abundance of false negatives in most models, this ensemble model also had a lot of false negatives.

3) Average Ensemble Model with Bias: To counter the issue of false negatives, a bias of $(0.1, 0.25)$ was added to the predictions by average ensemble before rounding them off. This allowed for more positives but was also not very reliable. Fine tuning it to 0.15 bias allowed for a reliable model however.

4) Weighted Ensemble Model: If one model performs significantly better than others in most cases, assigning it a higher weight helps better the final ensemble model. Weights can vary from it's score or the total number of models in ensemble. We use the second technique to ensure that the ensemble is *atleast* a certain model.

5) L2 Ensemble Model: Finally, a weighted ensemble was used where the weights was the activation value of that pixel itself. Hence this ensemble was dubbed *L2 Ensemble* as it was reminiscent of L2 regularisation.

IV. Conclusion

TABLE I
JACCARD SCORES FOR ALL MODELS

Model Name	**Training Score**	**Validation Score**
Model 1	0.769406	0.802823
Model 2	0.760612	0.816523
Model 3	0.877114	0.866433
Model 4	0.779381	0.895808
Model 5	0.893834	0.910138
Model 6	0.831964	0.804912
Model 7	0.846126	0.864094
Model 8	0.872854	0.922661
Model 9	0.907228	0.918812
Model 10	***0.929104***	0.919679
Model 11	0.917962	***0.959026***
Model 12	0.864672	0.903981

TABLE II
TEST JACCARD SCORES

Model Name	**Test Score**
Model 2	0.748298
Model 5	0.671013
Average Ensemble of Model 2/4/5/8/9 with 0.25 bias	0.754588
L2 Ensemble of Model 2/4/5/8/9	0.740317
Average Ensemble of Model 2/4/5/8/9/10/11 with 0.15 bias	0.745343
Weighted Ensemble of Model 2/9/10/11 with weight=4 for 2	0.755348
Weighted Ensemble of Model 2/9 with weight=2 for 2	0.75371
Average Ensemble of Model 2/9/11/12/12.1/12.2	***0.769539***

Due to limited number of attempts, we were unable to check the test scores for a lot of models and ensembles. The above scores are from the limited attempts given to us.

V. Acknowledgment

M.Z. thanks Dr. Prabhakar Shukla, Scientist , UKRI GCRF Water Security and Sustainable Development Hub, IIT Delhi for providing insights in visualising the data. We thank the HPC facility, IIT Delhi, for providing storage and computational resources.

VI. Open Source Software and python libraries used

- PyTorch [10]
- scikit-learn [11]
- matplotlib [12]
- imgaug [13]

References

[1] "Indian Geo Platform of ISRO," https://bhuvan.nrsc.gov.in/home/index.php.
[2] W. K. Bain and D. Bezbaruah, "Understanding Soanian occurrences at Bam locality of Siwalik frontal range, north-western India," *Anthropological Science*, vol. advpub, 2021.
[3] S. G. Patil and R. K. Lad, "Evaluation of Spatio-temporal Dynamics of Groundwater Recharge and locating Artificial Recharge Structures for Watershed in Upper Bhima Basin, Pune, India," *Journal of the Indian Society of Remote Sensing*, Jul. 2021.
[4] V. Sharma, M. Zaki, K. N. Jha, and N. M. A. Krishnan, "Machine learning-aided cost prediction and optimization in construction operations," *Engineering, Construction and Architectural Management*, vol. ahead-of-print, no. ahead-of-print, Jan. 2021.
[5] Jayadeva, H. Pant, M. Sharma, A. Dubey, S. Soman, S. Tripathi, S. Guruju, and N. Goalla, "Learning Neural Network Classifiers with Low Model Complexity," *arXiv:1707.09933 [cs]*, Mar. 2021.
[6] mateuszbuda, "U-net for brain segmentation," original-date: 2019-05-26T00:27:21Z. [Online]. Available: https://github.com/mateuszbuda/brain-segmentation-pytorch
[7] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-CNN." [Online]. Available: http://arxiv.org/abs/1703.06870
[8] D. Hong, L. Gao, J. Yao, B. Zhang, A. Plaza, and J. Chanussot, "Graph convolutional networks for hyperspectral image classification," vol. 59, no. 7, pp. 5966–5978, conference Name: IEEE Transactions on Geoscience and Remote Sensing.
[9] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation." [Online]. Available: http://arxiv.org/abs/1505.04597
[10] "PyTorch," https://www.pytorch.org.
[11] "Scikit-learn: Machine learning in Python — scikit-learn 0.24.2 documentation," https://scikit-learn.org/stable/.
[12] J. D. Hunter, "Matplotlib: A 2D Graphics Environment," *Computing in Science Engineering*, vol. 9, no. 3, pp. 90–95, May 2007.
[13] A. Jung, "Aleju/imgaug," May 2021.