# The Eye in The Sky

Aaryaan Sharma, Anand Uday Gokhale, Sumanth R Hegde

*Indian Institute of Technology Madras*

**Abstract**

Satellite image classification is an important task in order to extract relevant information from massive amounts of satellite imagery. There are various applications of satellite image classification, including spatial data mining, field surveys , disaster management , monitoring weather conditions,etc. As datasets grow bigger, it becomes harder to label satellite images by hand and it becomes more prone to human error. This gives rise to the need for a robust classification system for satellite images. In the given problem statement, the amount of training data was much lesser than any usual satellite imagery classification challenge. We propose a unique mix of classical vision and deep learning algorithms to tackle this classification task for the 8 classes - *Roads, Buildings, Trees, Grass, Bare Soil, Water, Railways and Swimming pools.*

## 1. Team Members

- **Aaryaan Sharma** (Contact number : 9731090474 ) A sophomore at IIT madras studying Chemical Engineering. As a part of the Computer vision and Intelligence Group, he has done a few projects involving computational photography, including an implementation of CNN-SLAM. His other interests include playing the guitar and gymming.
- **Anand Uday Gokhale** (Contact Number : 9980366443) A sophomore at IIT madras pursuing a dual degree in Electrical Engineering. As a part of the Computer vision and Intelligence Group, he has done a few projects, including an implementation of Deepfakes. Deepfakes is an image reconstruction problem attempting to morph expressions from one face to another. He has also implemented neural networks for detection of pneumonia in lung X-ray scans. His other interests include playing football and playing the keyboard.

- **Sumanth R Hegde** (Contact Number : 974057567) A sophomore at IIT Madras studying Electrical Engineering. Interested in the power of deep learning, he has worked on projects as part of the Computer Vision and Intelligence Group of IIT Madras, including an implementation of an attendance system based on facial recognition. His other interests include music and politics.

## 2. Classification Approach

### *2.1. Motivation*

The given training set consists of only 14 images. This is much lesser than any usual satellite imagery dataset, used in typical applications such as aerial surveying or city landscape segmentation. For a dataset of this size, there is sure chance that any deep learning model will over-fit the training dataset .Thus, our approach was to first exploit simple features in the images using classical computer vision. For classes that could not be segmented out with traditional computer vision, we implemented individual segmentation models, constrained by scale invariant and reconstruction regularization. Our main motivation to implement class-wise masks was because of the poor accuracy obtained in joint classification approaches. We exploit human observations on a variety of threshold, graph and histogram bounds in order to base our experiments and arrive at suitable methods for each class.

### *2.2. Pre-processing*

The satellite images in the dataset provided has various intensities and are not uniform. To ensure that the classifier works on all images it is necessary to ensure equal intensity on all of them. Hence we started off by performing *Histogram equalization* on all images on all 4 channels for contrast enhancement. We have implemented different deep learning models for which input was processed as 64x64 crops. In order to take care of boundary effects, we used padding with reflections, so as to also avoid errors during test time. More details of the pre-processing follows in section 2.5.1

### *2.3. Approach*

Although each image in the training dataset on average has a resolution of around 900x900, this data is too less to develop a robust deep learning algorithm that would not suffer from poor generalization by over-fitting the training data. Hence, we approached this with a **divide-and-rule** strategy by developing algorithms that would generate a binary classification mask for each class, using a mix of classical computer vision and deep learning. We then combine them to obtain the final segmented output. We elucidate this strategy for each class in the subsequent section.

## 3. Classwise Approach

Our classification strategies for each class are outlined below :

### *3.1. Water Bodies(Swimming pools and Water)*

Water has a unique property of absorbing a very large percentage of IR light that is incident on it. Hence water bodies appear black in the NIR band. Using the fact that this property is unique to water among the given classes, we were able to segment out the water bodies from the rest using simple thresholding. Next we proceeded to classify these bodies into swimming pools and water.

### *3.2. Water*

The NIR channel thresholded for water has a lot of noise due to shadows on the map. We try to minimise this noise by performing a pixel-wise *binary AND operation* on the image with a mask created by RGB thresholding to expel shadows and swimming pools. We then erode and dilate the final mask to:

- Supress fringe noise even further
- Eliminate obstacles like bridges so that we get a complete picture of the water body

To further improve accuracy, we use a contour area threshold to eliminate any large shadows that might have crept into the mask. Then we perform a pixel-wise binary AND operation on this mask with the first obtained NIR mask to get finer details like shores, bridges etc. to get our final water mask.

### *3.3. Swimming Pools*

Swimming pools have a characteristic color which makes them easier to distinguish from other water bodies and other noisy parts of the NIR band. Wherever there is a noisy section in other water bodies with a similar shade of blue we eliminated it by checking if the pixel was already classified as a member of the other water class.

### *3.4. Grass and Trees*

For segregating objects with chlorophyll, we used the CCCI map (Canopy Chlorophyll Content Index) which is calculated per-pixel by using the formula:

$$NDRE = \frac{NIR - RED}{NIR + RED} \tag{1}$$

where, $NIR$ is the pixel intensity array of the NIR channel , and $RED$ is the pixel intensity array of the red channel.

$$CCCI = \frac{(NDRE - NDRE_{min})}{(NDRE_{max} - NDRE_{max})} \tag{2}$$

Following this, we applied an appropriate threshold to create a mask containing only trees and grass. In order to differentiate between forests and grass, we used the following approach:

- We converted the RGB channels into the HSV colourspace. This is because trees are of a significantly darker shade than grass and we could exploit the hue value for classification.

- We equalized the value channel in the output from the previous step. The motivation for this was to equalize intensity among different images and also to ensure there is a significant difference between values for trees and grass.

- Applied an appropriate threshold to this image. This helped us separate grass and trees. The equalization done previously helped ensure that the value cutoff in different images is more or less the same.

### *3.5. Implementation details*

The final output is got by the combination of various class-wise masks implemented . The output map for each class is as follows - Water (dark blue), Swimming pool (light blue) , Trees (dark green), Grass (light green),Buildings (grey), Bare Soil (brown) ,Railways (yellow).

*3.6. Buildings, Roads, Railways and Bare soil:*

Building a classification mask for these set of classes is highly challenging, since traditional vision methods would fail to generalize for the wide range of colours a building could be in, and due to the inter-class similarity of pixel values. To address this problem, we built a unique combination of segmentation masks for each class.

Our general pipeline for each class and motivation for the same is as follows:

*3.6.1. Preprocessing*

The input images are equalized and broken down into patches of size 64x64. We chose this crop size so as to increase the batch size seen by the model as well as to preserve enough information in each crop for better segmentation. To incorporate boundary effects, we added zero padding. However, simply breaking the image into such 64x64 crops produced erroneous square edges during prediction time. We addressed this problem with the same method that is used in the U-Net Paper [1] , using reflections. Apart from this, to increase the amount of data we have we exploited the rotational and translational invariance by rotating and performing horizontal and vertical flips on the data provided.

*3.6.2. Model architecture*

We add Gaussian noise to the input patches of size 64x64 for added robustness and the patches are now fed to a Fully Convolutional Network (FCN). The FCN for each class has a modified U-net architecture. U-net is commonly used for a segmentation challenge constrained by less data . In order to have the best possible results on our training set, we have used the following two models:

- **U-Net architecture with more feature channels** *(U-Net (a))*: Given the input crop of just 64x64, we are forced to have less down-sampling layers for our model. Thus, to account for this, we used more feature channels at each block so as to increase the complexity of features learnt, along with batch normalization after every convolutional layer. To prevent overfitting, we added L2 regularization of 0.01 to the convolutional layers. We used the *He* [2] normal initialization, made popular by the ResNet paper

- **U-net (a) with more convolutional layers** (*U-Net (b)*) :
  In order to have more complex feature representation, we added additional convolutional layers to each block of the our previous *U-net* implementation(*U-net (a)*)) along with batch normalization and $L2$ regularization of 0.01.

Implementations such as Deep U-net [3] having more layers than the original U-net implementation have been found to perform better . However, the original Deep U-net model downsamples an input from size 1024x1024 to 8x8 , and these many downsampling blocks are too high for our model, due to the trade-off for sufficient batch size. We experimented with a Deep U-net architecture with less number of layers, but this performed worse than its U-net counterpart. Hence, we chose to use the U-net model with added modifications.

| Layer/Block | Output Shape | Parameters |
|---|---|---|
| input (InputLayer) | (None, 64, 64, 4) | 0 |
| gaussian-noise | (None, 64, 64, 4) | 0 |
| conv2d-1 | (None, 64, 64, 64) | 2368 |
| batch-normalization-1 | (None, 64, 64, 64) | 256 |
| max-pooling2d(1) | (None, 32, 32, 64) | 0 |
| conv2d-block-1 (3 conv +bn) | (None, 32, 32, 128) | 370560 |
| max-pooling2d-2 | (None, 16, 16, 128) | 0 |
| conv2d-block-2 (3 conv + bn) | (None, 16, 16, 256) | 1478400 |
| max-pooling2d-3 | (None, 8, 8, 256) | 0 |
| conv2d-9 ( 2 conv + bn) | (None, 8, 8, 512) | 3544064 |
| up-sampling2d-1 | (None, 16, 16, 512) | 0 |
| conv2d-11 | (None, 16, 16, 128) | 589952 |
| conv2d-block-2 + conv2d-11 | (None, 16, 16, 384) | 0 |
| conv2d-block-3 (3 conv bn) | (None, 16, 16, 256) | 1477120 |
| up-sampling2d-2 | (None, 32, 32, 256) | 0 |
| conv2d-14 | (None, 32, 32, 256) | 590080 |
| conv2d-block-1 + conv2d-14 | (None, 32, 32, 384) | 0 |
| conv2d-block-4 (3 conv bn) | (None, 32, 32, 128) | 739200 |
| up-sampling2d-3 | (None, 64, 64, 128) | 0 |
| conv2d-18 | (None, 64, 64, 128) | 147584 |
| batch-normalization-1+conv2d-18 | (None, 64, 64, 192) | 0 |
| conv2d-19 | (None, 64, 64, 64) | 110656 |
| batch-normalization-16 | (None, 64, 64, 64) | 256 |
| conv2d-block-5 (2 conv +bn) | (None, 64, 64, 32) | 27968 |
| conv2d-22 | (None, 64, 64, 1) | 289 |
| - | - | 9,078,753 |

**U-net(a) Architecture**

| Layer (type) | Output Shape | Parameters |
|---|---|---|
| input-1 (InputLayer) | (None, 64, 64, 4) | 0 |
| gaussian-noise-1 | (None, 64, 64, 4) | 0 |
| conv2d-block-1 (2 conv + bn) | (None, 64, 64, 64) | 39808 |
| max-pooling2d-1 | (None, 32, 32, 64) | 0 |
| conv2d-block2 (4 conv +bn) | (None, 32, 32, 128) | 517656 |
| max-pooling2d-2 | (None, 16, 16, 128) | 0 |
| conv2d-block-3( 4 covn +bn) | (None, 16, 16, 256) | 2069504 |
| max-pooling2d-3 | (None, 8, 8, 256) | 0 |
| conv2d-block-4 (3 conv +bn) | (None, 8, 8, 512) | 5905920 |
| up-sampling2d-1 | (None, 16, 16, 512) | 0 |
| conv2d-15 | (None, 16, 16, 128) | 589952 |
| conv2d-block-3 +conv2d-15 | (None, 16, 16, 384) | 0 |
| conv2d-block-5 (3 conv +bn) | (None, 16, 16, 256) | 2068224 |
| up-sampling2d-2 | (None, 32, 32, 256) | 0 |
| conv2d-19 | (None, 32, 32, 256) | 590080 |
| conv2d-block-2 + conv2d-19 | (None, 32, 32, 384) | 0 |
| conv2d-block-6 (4 conv+bn) | (None, 32, 32, 128) | 887296 |
| up-sampling2d-3 | (None, 64, 64, 128) | 0 |
| conv2d-24 (Conv2D) | (None, 64, 64, 128) | 147584 |
| conv2d-block-1+ conv2d-24 | (None, 64, 64, 192) | 0 |
| conv2d-block-7(2 conv +bn) | (None, 64, 64, 64) | 148096 |
| conv2d-block-8( 3 conv + bn) | (None, 64, 64, 32) | 37344 |
| conv2d-30 (Conv2D) | (None, 64, 64, 1) | 289 |
| - | - | 13,002,753 |

**U-net(b) Architecture**

*3.6.3. Model Optimization*

For each class, we experimented with standard loss metrics such as binary cross entropy (BCE) , dice loss, combination of dice loss and BCE, weighted binary cross entropy. The optimizer used was Nadam optimizer - Adam with Nesterov momentum for faster convergence.

We now explain the optimization strategy for each class in a categorical :
**Implementation for Buildings :**
For the buildings class, the ratio of positive examples (pixels belonging to this class) and negative examples (pixels not belonging to this class) is quite high in the training dataset. Thus, we found better results with *U-net(b)*,

a deeper implementation than the *U-net* . Out of total 4666 input crops of 64x64x4 size, we used 4128 for training and 538 for validation. Our experiments with loss functions are as follows:
Dice loss was found to give an train accuracy of 91% on this class, but due to class imbalance and similarity of certain road pixels with those of buildings, the results were riddled with false positives. So we experimented with *dice loss + BCE*. However, this too failed to eliminate false positives effectively. We got better results with simple binary cross entropy loss. To further improve our results, we went with a **weighted binary cross entropy loss**, and after grid based hyper-parameter search found an ideal weight for the positive targets term to be 0.6 . The model was trained for 100 epochs with a learning rate of $2e-4$. The train accuracy and validation accuracy were 88% and 86.9% respectively. Although less than the values dice loss, there were lesser false positives and thus a better kappa co-efficient.

**Implementation for Roads :**
For the roads class, the class imbalance is more than that of buildings, but still considerably less than railways or soil. Thus we went with *U-net(b)* architecture with a simple binary cross entropy loss trained over 50 epochs, at learning rate of 0.001. Out of the 4666 total patches, we used 4128 for training and 538 for validation. The train and validation accuracy were 89% and 88% respectively, reported at the epoch of minimum validation loss.

**Implementation for Railways :**
The training procedure for the railways class was more challenging. Our initial experimentation with same train and validation split as before and with dice loss resulted in accuracies of 99.11% and 99.37% at epoch of minimum validation loss. This was because the model learnt a black mask for everything! Due to the high class imbalance for railways, the model was able to achieve such high accuracies by simply classifying every pixel as not railways. We approached this problem with **hard mining**.

For hard mining, we consider any random point on an image and take a 64x64 crop, keeping that point as the top left point of the crop. If the number of pixels belonging to railways class is more than a set threshold, then it's a positive example. For each positive example, we also consider a negative example, where the crop does not contain any pixel belonging to this class. In our **final implementation**, we used a threshold of 15, so each positive example had at least 15 pixels of the railways class. We did this for each image that had pixel belonging to railways, and extracted 963 patches for training and 85 for validation. This is a very small number of input patches for the model, but we got better results than the previous approach. With class imbalance being taken care of, binary cross entropy loss performed better than dice loss, with model *U-net(b)* trained over 100 epochs and learning rate 0.001. Our training and validation accuracies were 94.87% and 93.86% at epoch of minimum validation loss.

**Implementation for Soil :**
This is another class where the number of positive examples were much lesser than negative examples. Thus we adopted a hard mining strategy similar to the railways class. With number of positive examples being lesser this time, we adopted a train and validation split of 888 and 40 crops. We sampled lesser number of crops because many of the generated random positive example patches would be very similar to each other, thus being reduntant. We used the *U-net(a)* architecture, in order to have sufficient model complexity as well as to prevent overfitting.We found the optimum loss function to be binary cross-entropy loss and we trained the model for 100 epochs with a learning rate of 0.001 . We achieved training and validation accuracies of 92.3% and 86.10%. respectively.

## 4. Results

Our final models are compiled as follows (extensive test outputs maybe found in the code base, we do not include them here for brevity) :

| Class | Architecture, Strategy | Epochs,Learning rate | Loss ,Optimizer |
|---|---|---|---|
| Buildings | U-net(b) | 100, 2e-4 | Weighted BCE, Nadam |
| Roads | U-net(b) | 100, 1e-3 | BCE , Nadam |
| Railways | U-net (b), hard mining | 100, 1e-4 | BCE, Nadam |
| Bare Soil | U-net(a), hard mining | 100 , 1e-3 | BCE , Nadam |

### *4.1.* ***Evaluation Metrics :***

For evaluation of metrics , we passed the input images in patches of 64x64 , with a stride of 32 so as to eliminate erroneous square edges formed at the

borders of each output crop. The average confusion matrix, kappa coefficient and overall accuracy are as follows :
Confusion matrix is reported with the following entries :

$$\begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix}$$

| Class | Kappa Coefficient | Confusion matrix | Overall Accuracy |
|---|---|---|---|
| Buildings | 0.757 | $\mathrm{a}_{00} = 1919229, a_{01} = 93713, a_{10} = 105377, a_{11} = 473321$ | 92.3% |
| Railways | 0.7 | $\mathrm{a}_{00} = 2554108, a_{01} = 7122.71, a_{10} = 11588.42, a_{11} = 18821.71$ | 99.27% |
| Roads | 0.784 | $\mathrm{a}_{00} = 1057311, a_{01} = 32878, a_{10} = 41500, a_{11} = 164131$ | 94.2% |
| Water | 0.687 | $\mathrm{a}_{00} = 878128, a_{01} = 8009, a_{10} = 198, a_{11} = 121728$ | 99.18% |
| Swimming Pool | 0.608 | $\mathrm{a}_{00} = 1437039a_{01} = 93713, a_{10} = 105377, a_{11} = 473321$ | 99.18% |
| Trees | 0.66 | $\mathrm{a}_{00} = 1156171, a_{01} = 33484, a_{10} = 29456, a11 = 76710$ | 95.14% |
| Grass | 0.45 | $\mathrm{a}_{00} = 1125133, a01 = 25988, a_{10} = 81498, a_{11} = 63201$ | 91.7% |
| Bare soil | 0.49 | $\mathrm{a}_{00} = 1985395.22, a_{01} = 3635.33a_{10} = 11516.11, a_{11} = 15174$ | 99.24% |

## 5. Conclusion

Satellite imagery segmentation is usually done on large amounts of data. Given the low amount of data, we have opted a unique approach using traditional computer vision, exploiting simple features of the satellite images and extensive deep learning experimentation, in order to use more complex features. Our model is built such that it would generalize as much as possible on new data while also performing effectively on the training data.

## 6. References

[1] O. Ronneberger, P. Fischer, T. Brox, U-Net: Convolutional Networks for Biomedical Image Segmentation, arXiv e-prints (2015) arXiv:1505.04597.

[2] K. He, X. Zhang, S. Ren, J. Sun, Deep Residual Learning for Image Recognition, arXiv e-prints (2015) arXiv:1512.03385.

[3] R. Li, W. Liu, L. Yang, S. Sun, W. Hu, F. Zhang, W. Li, DeepUNet: A Deep Fully Convolutional Network for Pixel-level Sea-Land Segmentation, arXiv e-prints (2017) arXiv:1709.00201.