Boundary detection for Field Delineation from GeoTIFF Images Using Neural Networks

THESIS SUBMITTED TO

SYMBIOSIS INSTITUTE OF GEOINFORMATICS

FOR PARTIAL FULFILLMENT OF THE M.Sc DEGREE

BY

ARBAZ SAYED

(BATCH 2021-2023)

M.Sc. Data Science and Spatial Analytics

Symbiosis Institute of Geoinformatics

Symbiosis International (Deemed University)

5th Floor, Atur Center, Gokhale Cross Road, Model Colony, Pune – 411016, Maharashtra, India.

CERTIFICATE

Certified that this thesis titled **'Boundary detection for Field Delineation from GeoTIFF Images Using Neural Networks'** is a bonafide work done by Mr ARBAZ SAYED, at Hardcastle Agrotech Solutions Pvt. Lmt, Pune and Symbiosis Institute of Geoinformatics, under our supervision.

Supervisor, Internal

Dr. Vidya Patkar

Deputy Director

Symbiosis Institute of Geoinformatics

ACKNOWLEDGEMENT

This project would have not come to success without my external guide, **Mr. Neil Jadhav (CEO, MapMyCrop)** who allowed me to work as an Intern in his company Hardcastle GIS. He gave a helping hand in attaining cloud services and resources needed for this project.

Special thanks to my project mentor, **Mr. Ashutosh Pawar (Data Scientist MapMyCrop)** who was there constantly throughout the completion of my project and helped with all the technical issues that I countered through this journey.

I would like to specially highlight the efforts given by my internal guide, **Dr. Vidya Patkar (Deputy Director SIG)** who gave a proper path to this project and was responsible for motivating me to complete the milestones of the project. She also helped me understand some of the challenging concepts which were required to make the project come to action.

INDEX

List of Figures

List of Tables

Abbreviation List

- GT - Ground Truth
- NN - Neural Network
- UI - User Interface
- FCN - Fully connected network
- ReLU - Rectified Linear Unit
- Adam - Adaptive Momentum Optimizer

PREFACE

This report is a constituent of the successful research conducted in the area of applying neural networks for the effective delineation of field boundaries. However, neural networks have not been exclusively tuned for this particular application. An attempt has been made for applying the neural networks which are specific to the image segmentation problems for getting the best results after tuning the hyperparameters and using the best possible data for training the model.

The report contains a brief explanation of what neural networks are and touches on the important concepts which are essential for optimizing the network. Then the topic revolves around the important task of increasing the accuracy of the network which in turn is a great subject of interest as many techniques are required for the analysis and understanding of the different branches within a network.

The latter part will explore deeply the results obtained and which technique, parameters, and hyperparameters led to the most accurate prediction. The results obtained will be justified based on statistical grounds accompanied by various graphs.

Even though, not much research has been yet initiated or has been completed to compare the results to a large sample of model standards. Still, the model built by me has managed to get one of the highest accuracies in this area of knowledge.

CHAPTER - I
INTRODUCTION

Boundary detection for Field Delineation from GeoTIFF Images Using Neural Networks

1. Introduction

An important step in various agriculture-related remotely sensed pipelines is the precise segmentation of huge amounts of land. Using a state-of-the-art deep learning algorithm with nothing more than a singular Sentinel-2 picture as feed, this research attempts to entirely simplify this time-consuming and resource-intensive operation. Identifying the boundaries that separate one cropland from the other on the field has been one of the most challenging tasks due to many obstacles such as cloud cover, the gap between machine learning techniques and satellite images, and spatial-temporal limitations of satellite data. GeoTIFF images are raster image files that store the satellite or drone mapped images and retain their quality even after they are compressed or manipulated which is ideal for working with imagery data for this matter. For bridging the gap between earth observation data and the current machine and deep learning techniques EO learn (Earth Observation) library was introduced which contains various pre-defined tasks for cloud masking, raster image loading, etc. The boundaries will help to identify the different pastures of land adjacent to each other which may or may not have similar or dissimilar crop types.

1.1 Significance of the Study:

Many agricultural operators need a precise understanding of field borders. This is a required resource for farm owners to onboard farms on modern farming software programs, it enhances multiple cropping categorization accuracies (Francois Waldnera, Foivos Diakogiannis, 2019), and it is used by government agencies to track incentives and food production, along with many other uses. The awareness of location of fields on the farms is of great importance to a farmer. Even though this can be done for small farm holders by manually surveying the field but it is a very human resource-intensive task for huge farms. Knowing the partition of the fields by their serial

number or name can significantly improve the management of fields especially when it is reaping season. Also, the boundary shapefiles can be integrated with a precision farming platform through which farmers can keep a track of all the activities taking place in all their fields, which was not possible before. Some organizations do the time-intensive task of creating the fields manually on Geoprocessing software which takes hours and hours for huge farm settings. This research work can reduce the time required for the same process by almost 95 percent and give a result that is 96 percent similar to the manually drawn fields.

The development, operation, and management of food security policies are hampered by the lack of full spatial details relating to agricultural areas. One of the primary pieces of data required to calculate agricultural output at the regional or national level is an agricultural area. A reliable agriculture monitoring system may be implemented with the use of Earth Observation data. A satellite-based method can be much more cost-effective and efficient than conventional field surveys, which makes it possible to systematically map land resources across vast geographic regions in other places around the world. Using photos with Very High Spatial Resolution (VHR), vast geographic regions may be mapped. However, due to the features of farmlands in some fields, which include a plot area of just approximately 2 hectares, precisely mapping agricultural resources is a difficult undertaking.

1.2 Aim and Objectives:

The focus will be to figure out an effective and viable way for masking out the clouds from satellite images and identifying the boundaries of the fields and segmenting them accordingly for accurate identification of croplands. Some of the important goals which will be the locus of this research work are:

- To achieve the optimal semantic segmentation of field boundaries by training a neural network.
- To set up an end-to-end pipeline that would convert the raw GeoTIFF images to field boundary polygon shapefiles.
- To set up a benchmark in accurate boundary extraction from earth observation images.
- To help farmers achieve sustainable farming practices.

1.3 Problem Statement:

How to effectively extract boundaries from raw satellite images of farms and convert them to polygon shapefiles of fields for precision farming?

To overcome this problem, a python notebook is created with all the necessary documentation needed for uploading the data and utilizing the trained model with its respective weights. Then a step-by-step approach is given for converting the neural networks' outputted field boundaries into shapefiles.

CHAPTER - II
LITERATURE REVIEW

2. Literature Review:

Literature Review is conducted for a better understanding of the domain that is being worked upon and to figure out the gaps that are needed to be filled in the area of research that is being conducted. It is employed not only for finding main ideologies conclusions, and theories but also to establish similarities & differences between the various works conducted. It also makes the researcher more aware of the domain knowledge that is to be researched.

For this research, a Literature review plays an important role in finding out the traditional computational techniques and machine learning methodologies that may be used to classify the extensive research on field border detection methods. The literature about the research conducted in this field of study will give important insights into which technologies were used for this task and how much accurate were the aggregate predictions. The extensive review gave valuable information about the technologies that were not used for this task and which ends they did not meet. The literature that was found most relevant for this work is cited below:

Franc¸ois Waldnera et al. (2019) approached the boundary detection problem with a very robust approach but did not account for overfitting. They successfully identified the croplands but didn't classify them based on crop types. In circumstances when the class-based pixel frequency is smaller than the visual span, AgriSegNet's (Tanmay Anand et al., 2021) multi-scale attention-based learning offers correct predictions. Their model received a 67 % F1 score.

Two fractal ResUNet models were developed in (Ruoyu Yang et al., 2020). One is based on information from Southern Africa (Source to Target), and the other is based on data from Australia (Target to Target). After 255 timesteps for the classification model on data from Southern Africa (MCC = 0.85) and 163 epochs for the method based on data from Australia (MCC = 0.87), optimal activation was reached.

For example, Foivos I. Diakogiannis et al. (2021) used a computer vision-based technique to compute field boundaries based on 3 and 4-band NIR imageries. The 4-channel imaging gave them a somewhat better outcome than the 3-channel imagery.

Iasonas Kokkinos et al. (2016) combined their model with an FCNN model based on semantic segmentation. The results of multi-resolution processing with tied-weights and late score fusion

were much superior to those of a single-resolution network. They pushed the bounds of boundary detection by allowing the boundaries detector to simply use intuitive grouping signals like closedness or consistency, which may frequently result in gains in the elevated domain.

Rishi Kumar Suresh Kumar (2020) got overlapping boundaries and boundaries incorrectly classified due to the same crop types existing next to each other. Crop bounds for standard machine learning models were not optimized quite well in the model, but the model appears to predict well even in image chip areas with poor crop occupancy, implying that the model can distinguish crop sites even when there is a lot of noise around them.

In Saining Xie, Zhuowen Tu's (2015) method, the author developed Holistically-Nested Edge Detection, which comprises three properties: automatic image learning (property 1), multi-scale response merging (property 2), and potential involvement of multiple levels of visual awareness (property 3). They claim that in the higher recognition regime, Heuristic Edge Detection outperforms other deep learning-based approaches. By recycling the poor prediction data, they were able to reduce the training time.

Alireza Taravat et al. (2021) proposed a model which can predict the field boundaries by overlooking the city areas and forest areas. They found no evidence of overfitting of data by training on the CNN architecture. In some areas, the model even corrected the ground truth data.

Recent developments in field boundary recognition have included U-Net topologies. Using a U-Net architecture, Surabhi Lindwall et al. (2022) classify images under 3 groups: fields, buffered borders, and backdrops. Considering the scale and gathering restrictions placed on self-constructed samples, they excel at edge detection of field borders. In comparison to the U-Net model, which has an average F1 score of 0.88 and an average Jaccard coefficient of 0.78, the ResU-Net framework has outstanding quality with a mean F1 score of 0.91 and a mean Jaccard coefficient of 0.88.

In this report, a method is introduced that is intended to be the initial stage in a process for consistent field border detection. The next part provides a detailed explanation of this method, which is created to eliminate scalability-related difficulties.

CHAPTER - III
STUDY AREA

3. Study Area Location:

The study area is chosen as the Netherlands due to the availability of precise fields which was expected to help train the neural network. Not only the fields from the Netherlands were found best organized for field delineation training but also the ground truth was available for each field which was more crucial for training the model.

In the figure below, the distribution of fields is shown on the map of the Netherlands:

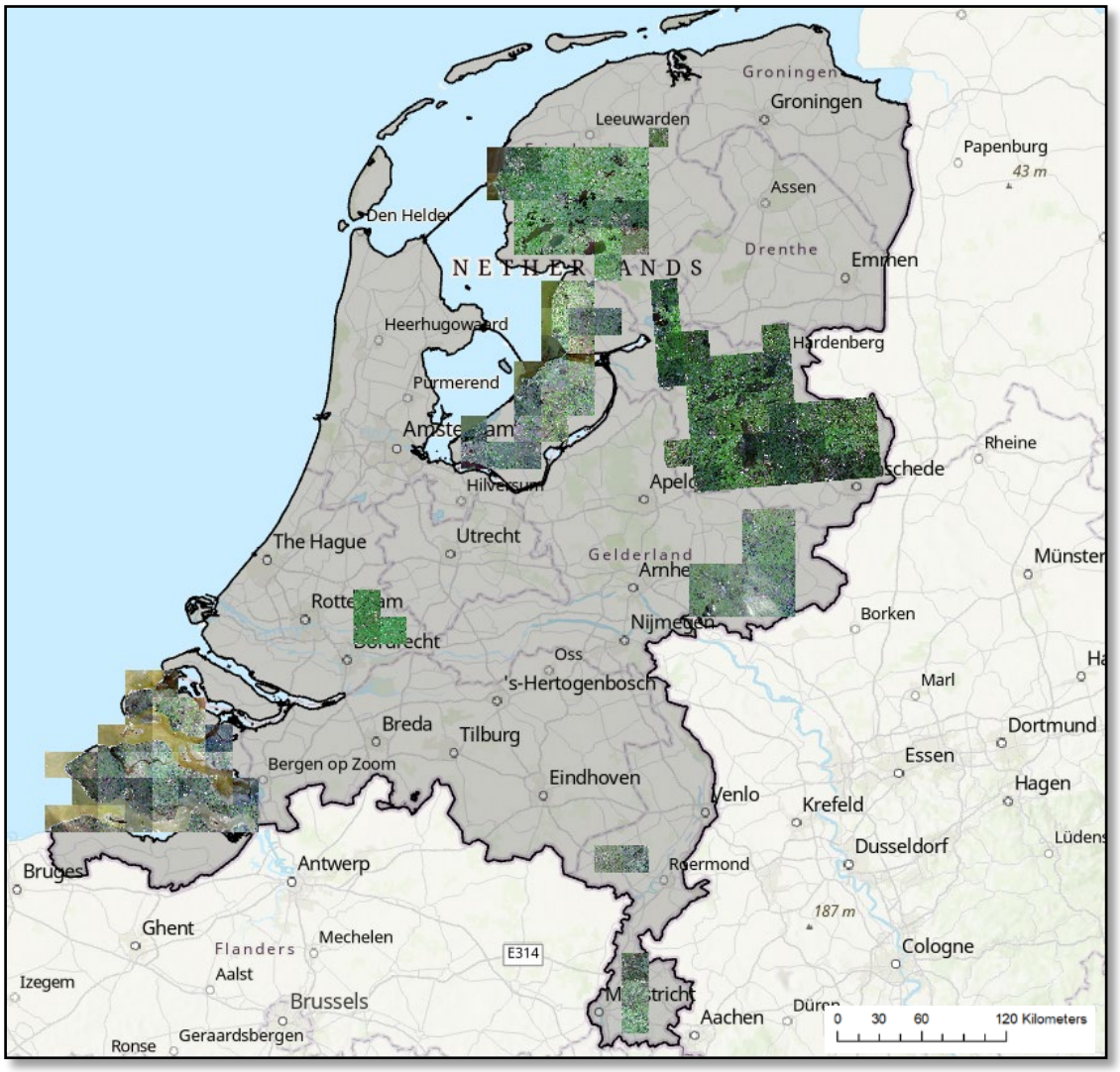


Figure 1: Distribution of field GeoTIFF patches in the Netherlands (Image processed on ArcGIS pro)

Around 122 image tiles are utilized which are distributed according to the abundance of fields.

In figure 2, we can see the area-wise distribution of the provinces in the Netherlands and the image tiles encompassed in each of them, the whiter regions represent larger provinces and darker ones represent the smaller ones. The most amount of image tiles come from Overjissel i.e., 36 tiles, and the least number of tiles come from Gelderland i.e., 3.

The largest Area is of Gelderland with 5118.59 sqkm and the smallest area is of Utrecht with 1554.11 sqkm.

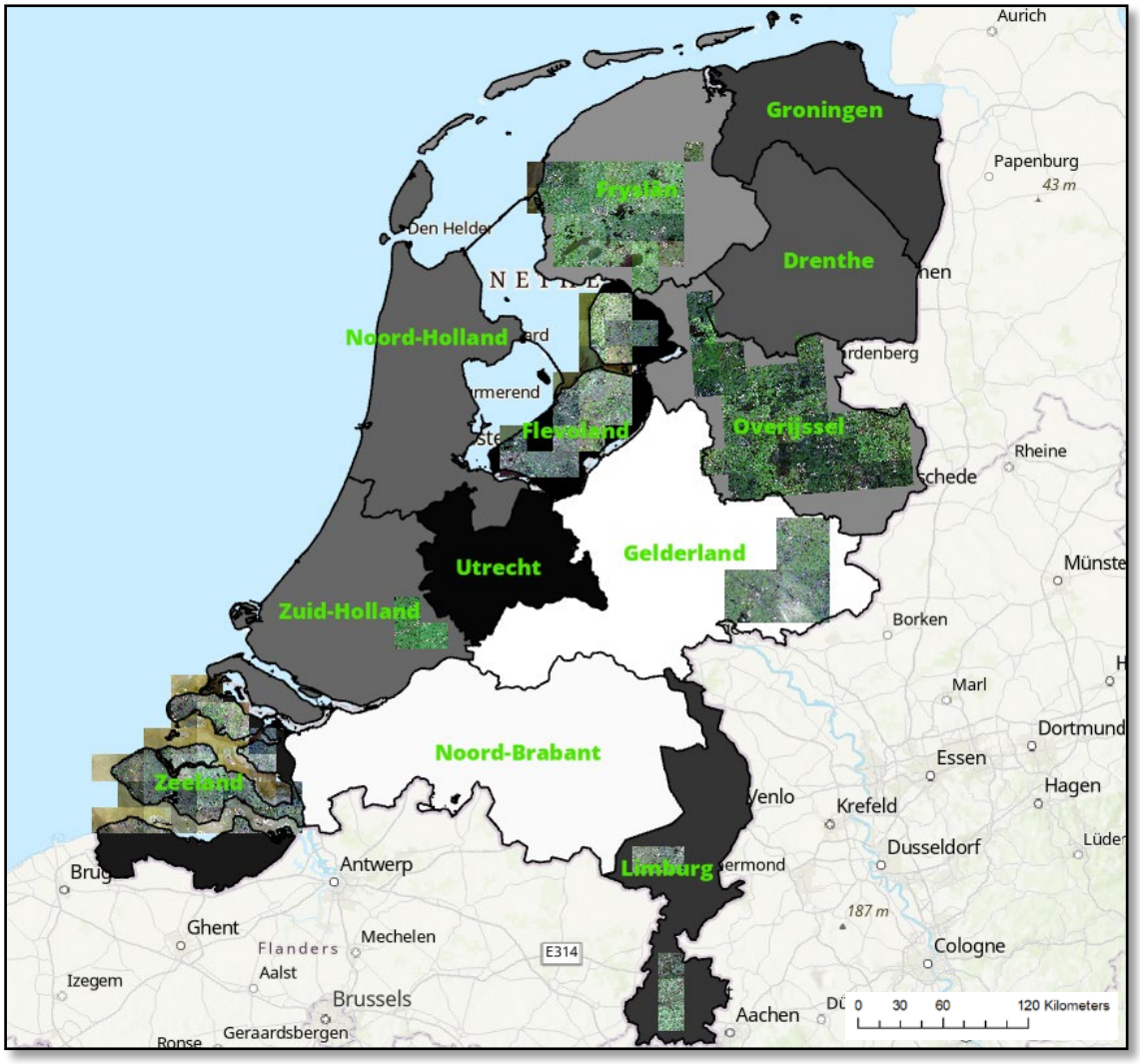


Figure 2: Netherland provinces area wise distribution with image tiles overlayed

CHAPTER - IV
METHODOLOGY

4. Methodology:

4.1 Software Used:

- The high-end GPUs provided by the Google engine backend through the **Colab UI** which uses python as the primary language was used for preprocessing, training, and saving the neural networks.
- ArcGIS Pro 3.0.0.3 was used for data post-processing.
- Microsoft Office Word 2022

4.2 Workflow Diagram:

A flowchart diagram of working helps in a better understanding of how the workflow is going. It also helps to get a clear, better picture or visualization of the work process. All the steps of the work processes of this thesis paper are shown in proper order.

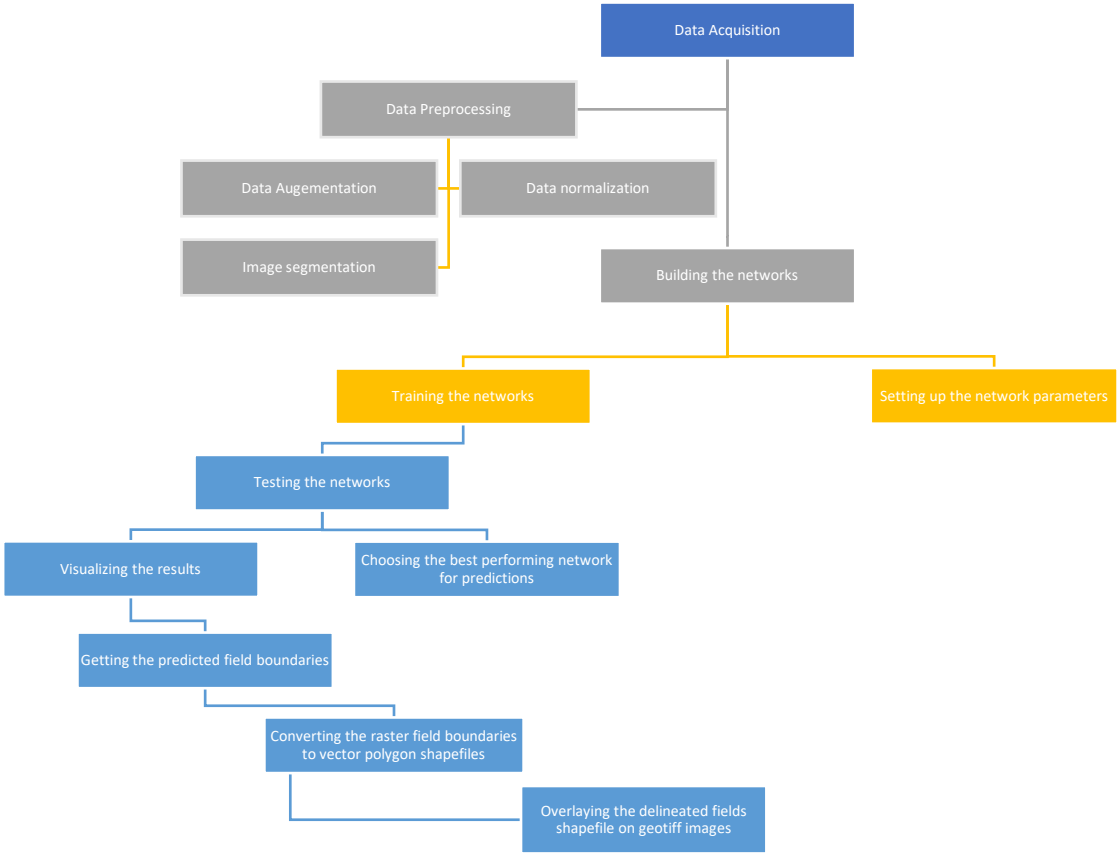


Figure 3: Workflow diagram

4.3 Data Source and Acquisition:

The Sentinel 2 data for the different provinces in the Netherlands was collected from the Sentinel Hub EO browser, link. The data was ideal for training the Fully convolutional neural network as it had very little or no cloud cover. Now, for the neural network to learn which pixels represent the ground truth data, precise ground truth data was required which was acquired from the Copernicus open-source project by the ESA, link. Both the data are in GeoTIFF format which is georeferenced and hence can be overlaid by using geoprocessing software. The figures below represent a sample of images and their corresponding ground truths. (Fig 4 – 6)

4.4 Data Preprocessing:

a. Converting Images into Pixel-Wise Arrays:

The GeoTIFFs of original images are first converted into pixel-wise arrays which are in 3 dimensions, the first and second dimension contains the number of x rows and y columns, and the third dimension contains the bands of images i.e., RGB and NIR. Classification images or the ground truth tiffs are converted into a 2d array.

But the per pixel values of the 2d array of the classification image are 255 and 0 for boundary lines and non-boundary respectively. These values are replaced by 2 for boundary and 1 for no boundary.

b. Normalizing the image:

Using the Normalized Digital Vegetation Index, the pixel values of the original image have been normalized.

Figure 4-a: Friesland 123 Original

Figure 4-b: Friesland 123 GT

Figure 4-c: Friesland 123 Overlayed

Figure 5-a: Flevoland 20 Original

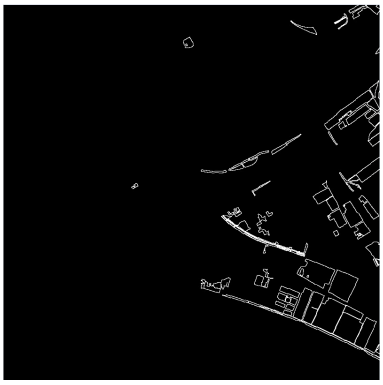


Figure 5-b: Flevoland 20 GT

Figure -c: Flevoland 20 Overlayed

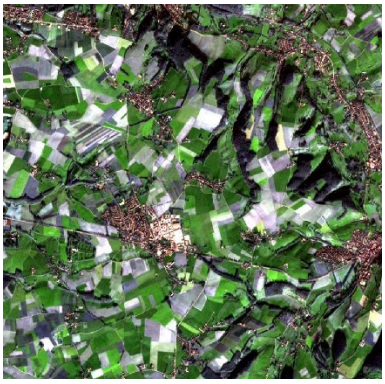


Figure 6-a: Limburg 82 Original

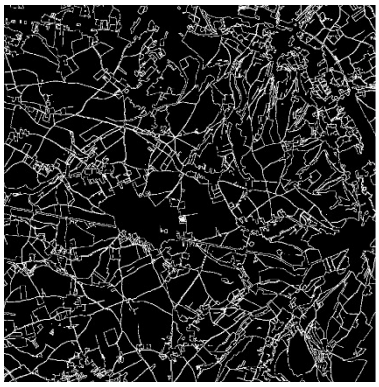


Figure 6-b: Limburg 82 GT

Figure 6-c: Limburg 82 Overlayed

c. Dividing the images into smaller patches:

The patches are made for the more efficient training of the network, and faster processing. The patch size depends on the network being trained. For e.g. Feeding a smaller patch size, ideally of 50px * 50 px is preferred whereas for UNet 2 a patch size of 200px x 200px is biased.

d. Setting up training and testing data set:

A random generator with seed 34 is used to split the data into a training and testing set of original images and ground truth data such that at least one image patch is included from each area in the testing set.

4.5 FCN model development:

FCNs or Fully Connected Networks have recently made notable accomplishments in pixel-wise segmentation. The preprocessing was done in conformity with the neural network's design. FCNs are made to automatically deduce pixel-wise estimates, irrespective of the quality of the source images, as opposed to typical CNNs, that forecast one classifier per source image. Wavelet or unpooling layers, which sample the local features discovered by the fully connected layers to the resolution of the input picture in the upward direction, typically replace the convolution operation of traditional CNNs in these systems. Utilizing "no down-sampling" systems with dilated convnet is an alternate strategy. Moreover, FCN models can be tuned to perform the task of Semantic Segmentation and can learn at a faster rate. For pixel-wise classification, FCNs are made up of a deep codec. The network's encoder, which consists of many convolutional layers, batch normalization, and Rectified Linear Units, is ontologically similar to the convolutional layers of the VGG-16 network.

The decoder would be used to convert the encoder's reduced feature maps toward the input image's native resolution. The decoder performs quasi-up sampling utilizing grouping values obtained in the relevant max-pooling levels of the sequencer rather than discretization or inverted repetitions. The resultant up-sampled maps are sparse, therefore robust component maps are created by convolving them using trainable convolution kernels. By doing away with learning to upsample, this approach decreases the number of trainable parameters while increasing border demarcation accuracy. The FCNs are an effective network for the job of segmentation to these features. The basic foundation of FCN can be illustrated in the figure shown beside:

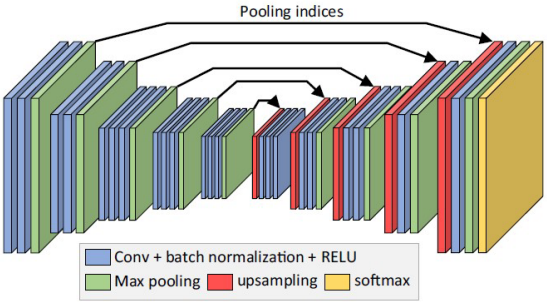


Figure 7: Fully convolutional network archtiecture

4.6 UNet Model Development:

Unet was originally developed and designed for the task of processing biomedical images but slowly, its potential for various other purposes was also recognized. In the topic of interest, Unet's architecture is promising for completing the task of field boundary delineation as it has given far more than satisfactory results in object boundary detection. One of the major advantages of UNet over FCN is its ability to detect features from a low-resolution image and give good results. UNet can work with a larger sample size at a time rather than dividing the samples into smaller sizes while training. The picture depicts how a UNet's entire shape resembles a letter U.

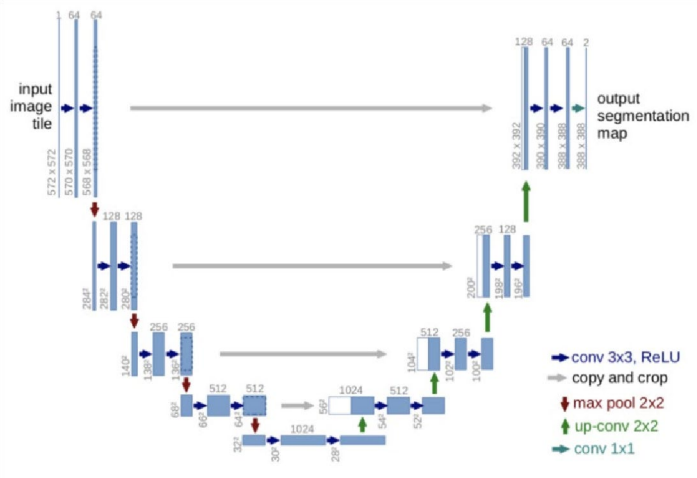


Figure 8: Unet Model Structure

The pattern is essentially the same from each side. Just on left, you can see the down-sampling, or contraction, process. On the right are the transposed convolutions and the up-sampling path. As the convolution process increases the complexity of the image, you'll see each step comprises of 2 convolutional layers and also that the number of channels increases from One to Sixty-Four. The maximum pooling procedure, indicated by the red arrow pointing downward, decreases the pixel count of the picture by half. The procedure is then done three more times. At 28x28x1024, the picture has been resized.

Now, speaking about the expansive path, the basic process is as follows:
(Convolutional 2d transpose -> Concatenate ->Convolutional layer 1 -> Convolutional layer 2) If the image while upsampling is not in the optimal size then transposed convolution performs some padding to the image. The picture is upsized from 56x56x512 following the transposed convolution. This picture is combined or concatenated with the comparable image from the contracted route, and the combined result is a picture with the dimensions 56x56x1024. Next, the data from earlier layers are combined to produce a more reliable prediction.

4.7.a Activation Function - Leaky ReLU:

Leaky ReLU which stands for Leaky Rectified Linear Unit was proposed to solve the problem of dying neurons. In cases in which the weighted total of inputs is negative, it adds a little slope to maintain a constant update. It's described as: $f(x) = \max(0.01a, a) = \max(0.01a, \sum_{i=1}^{i=n} w_i x_i + b)$

4.7.b Activation Function – Softmax Activation function:

The activation function known as SoftMax transforms integers and logits toward likelihood. A SoftMax produces a matrix (let's say v) with probability for each potential result. In vector v, the chances add up to one for all possibilities or categories.

By taking the coefficients of every outcome and normalizing each value by the average of such exponents, SoftMax converts logits (the numerical output of the final linear layer of an inter-classification neural network) into odds. As a result, all probabilities should add up to one. The typical loss function for such several co-classification tasks is cross-entropy loss. Typically, the last layer of an image base classifier, like in CNN, will have SoftMax included.

4.8 Network Optimizer – Adam:

The Adam optimizer uses momentum as well as a dynamic learning rate which optimizes the learning rate even further when it reaches the local infimum. To discover unique learning rates for each parameter, the algorithm makes use of the power of adaptive learning rates approaches. It also benefits from Ada-grad, which excels in sparse gradation situations but suffers with quasi optimization of neural nets, and RMSprop, which attempts to address most of Adagrad's issues and performs admirably. Adam can be looked upon as an algorithm that stands on the shoulders of Stochastic Gradient Descent and RMSprop which themselves are good in optimizing a network.

4.9 Choosing the parameters of the model:

The most important parameter to choose while building a model is the number of layers we want to insert into the network. For this project, FCN was tested with layers 3,4,5, and 6, and unet was tested with layers 3 which is claimed to the optimal number of layers for the image segmentation tasks.

Each network is given a separate UUID so that the model history and trained weights can be accessed easily after the model has finished training. This way, retraining of the model is avoided and speedy prediction is achieved.

Some of the other important parameters are batch size, epochs, Adam's learning rate, Adam Beta 1, Adam Beta 2, and Adam Epsilon. The batch size determines how much data is to be passed in a particular batch at a time to the network for training, an epoch is the number of passes of data we want to make through the layers of the network and the latter is hyperparameters which determine the behavior of Adam optimizer.

Each parameter was chosen by a mix of strategies, which involved tweaking the standardized parameters for the use case of this study and by training the model on the range of parameters which are estimated by calculation to give the best results.

CHAPTER - V
RESULTS AND DISCUSSION

5. Results:

The networks built were tested across a variety of metrics, also the networks were tested for a few hyperparameters for which the model was expected to give the best results. Out of these, the best networks with the optimal value of hyperparameters were taken forward for training for a higher number of epochs.

The results are divided into different sections which follow a waterfall approach for a better understanding of how this approach arrived at the best possible outcome of field delineation.

5.1 Network parameters:

The network configurations are mentioned in the table below:

Network Name	Epochs	Batch Size	Patch Size	Learning Rate
Fcn 5	50	8	40000 px	0.1
	800	32	2500 px	0.001
Fcn 6	50	8	40000 px	0.1
	800	32	2500 px	0.001
Unet 2	50	8	40000 px	0.1
	800	8	160000 px	0.0013
Unet 3	800	8	160000 px	0.0013

Table 1: Network parameters

The patch sizes are adjusted in direct proportion to the number of layers present in a neural network It is important to note here that 50 epochs were a dry run for testing which hyperparameter was optimal for a particular network. After which the training was run for 800 epochs to get the global minima in the loss function.

5.2 Network performance visualization:

All the visualizations are for 800 epochs

a. *Fully Connected model with 5 layers:*

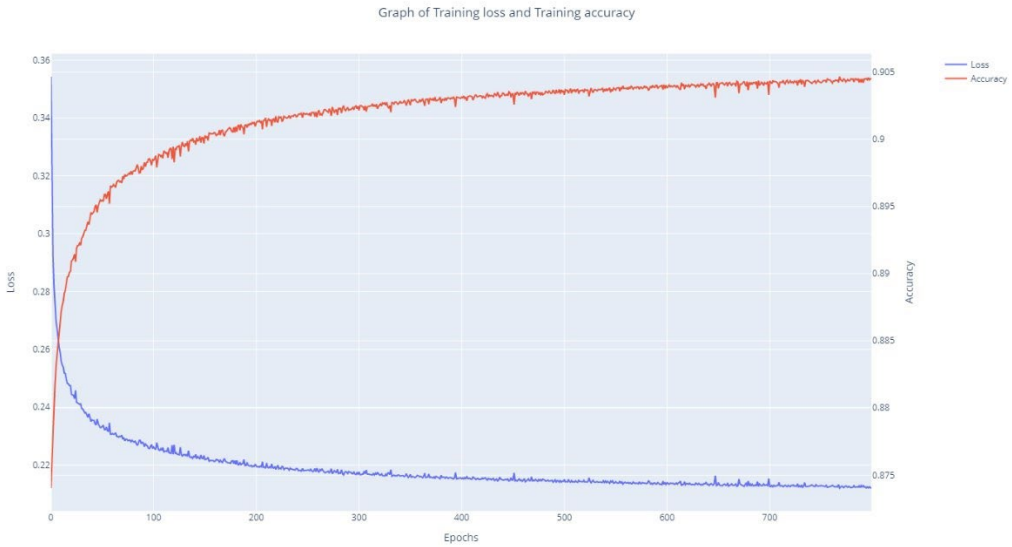


Figure 9: FCN (5 layers) – Training loss and training accuracy VS epochs

The graph depicts the increase in training accuracy and a corresponding decrease in training loss with the increasing number of epochs fully convolutional neural network with 5 layers. The training accuracy obtained at the 800th epoch is around 0.89 percent and the lowest loss that was attained is around 0.11 percent.

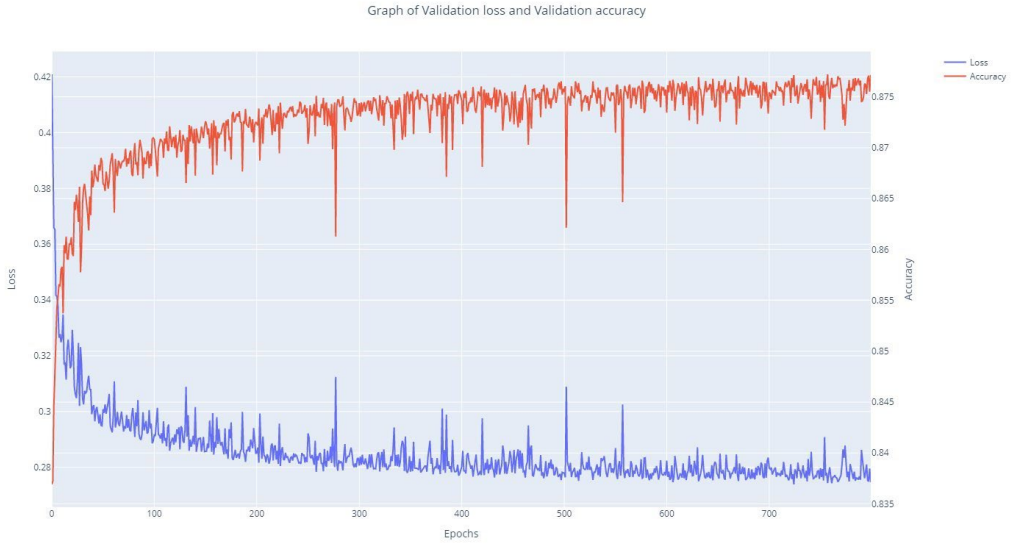


Figure 10: FCN (5 layers) – Validation loss and Validation accuracy VS epochs

The graph depicts the increase in validation accuracy and a corresponding decrease in validation loss with the increasing number of epochs for a fully convolutional neural network with 5 layers. The training accuracy obtained at the 800th epoch is around 0.88 percent and the lowest loss that was attained is around 0.12 percent.

b. Fully Connected model with 6 layers:

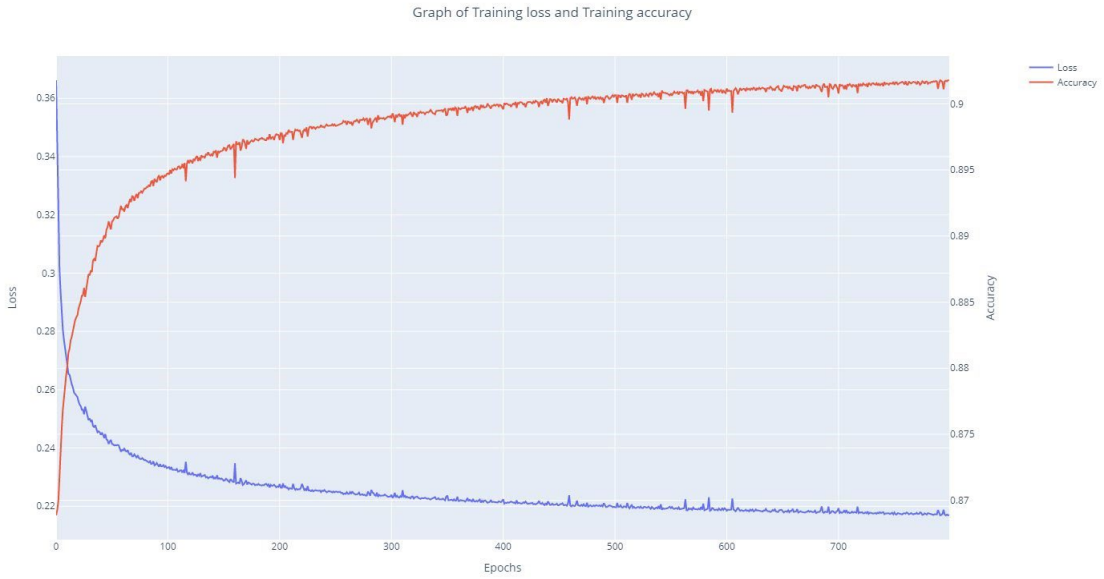


Figure11: FCN (6 layers) Training loss and training accuracy VS epochs

The graph depicts the increase in training accuracy and a corresponding decrease in training loss with the increasing number of epochs fully convolutional neural network with layers. The training accuracy obtained at the 800th epoch is around 0.91 percent and the lowest loss that was attained is around 0.09 percent.

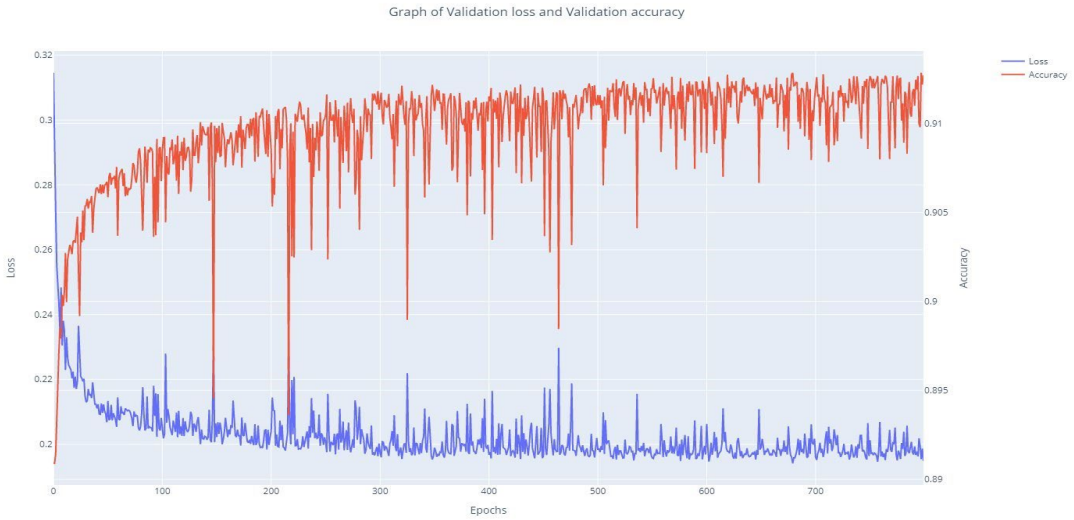


Figure 12: FCN (6 layers) – Validation loss and Validation accuracy VS epochs

The validation accuracy obtained at the 800th epoch is around 0.923 percent and the corresponding validation loss that was attained is around 0.09 percent.

c. Unet 2 with 800 epochs:

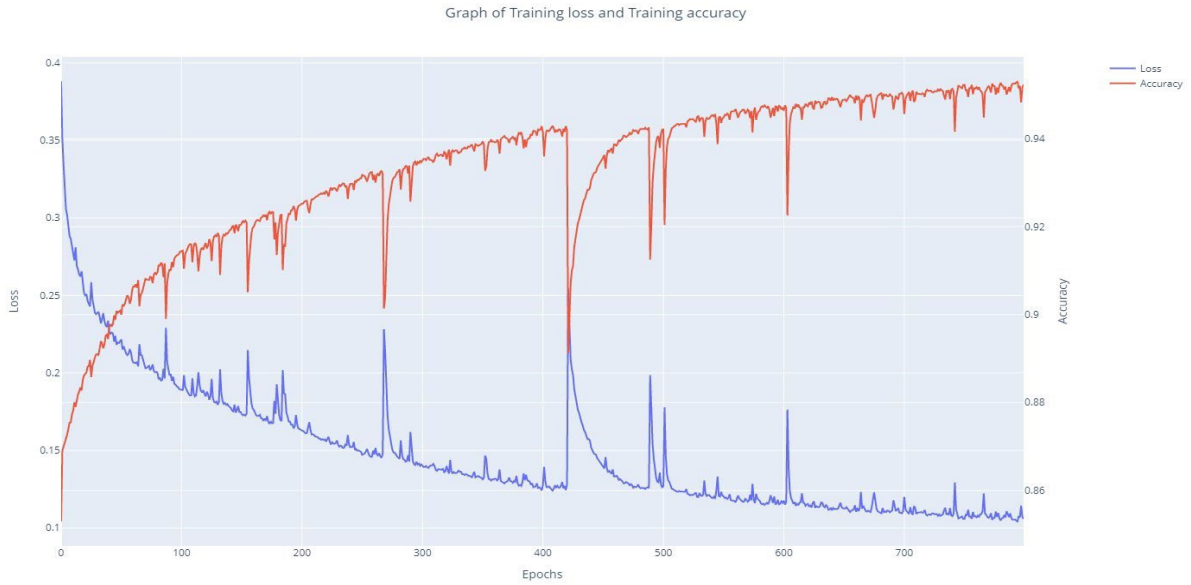


Figure 13: Unet 2 – Training loss and training accuracy VS epochs

The graph depicts the increase in training accuracy and a corresponding decrease in training loss with the increasing number of epochs fully convolutional neural network with layers. The training accuracy obtained at the 800th epoch is around 0.95 percent and the lowest loss that was attained is around 0.1 percent.

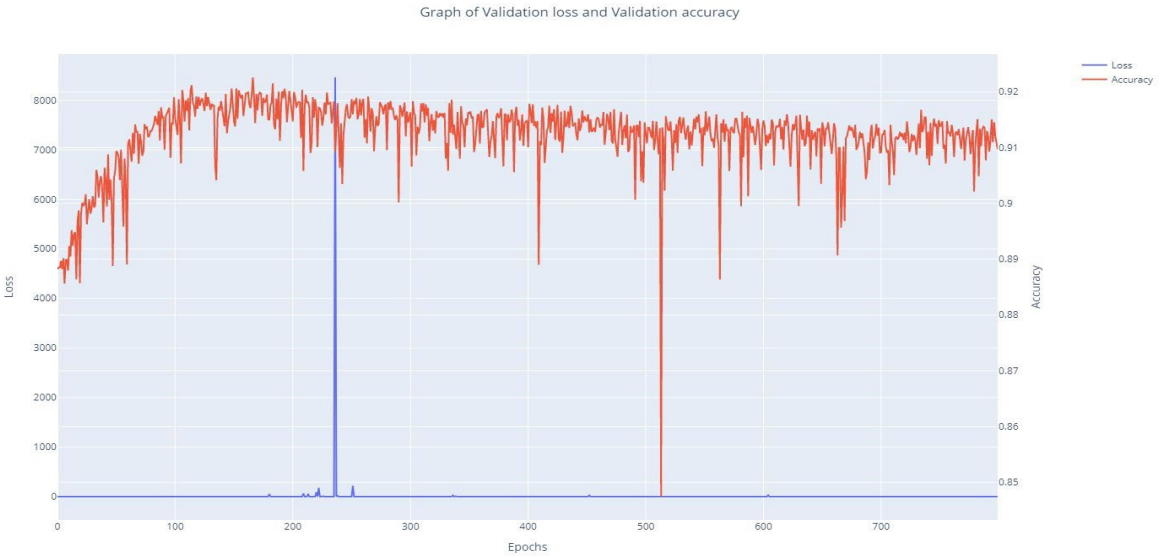


Figure 14: Unet2 – Validation loss and validation accuracy VS epochs

The validation accuracy obtained at the 800th epoch is around 0.915 percent and the corresponding validation loss that was attained is around 0.01 percent.

d. Unet 3 with 800 epochs:

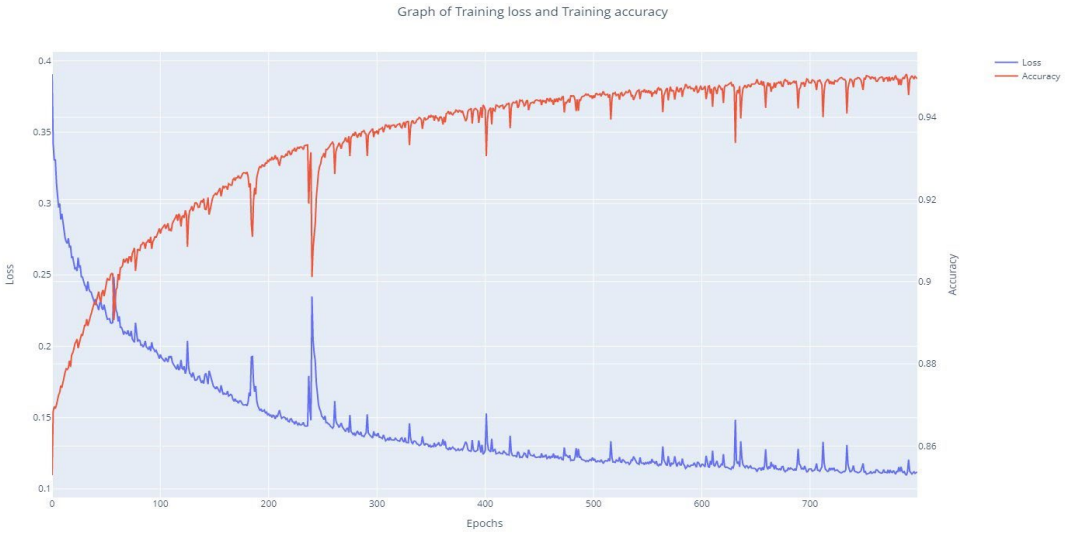


Figure 15: Unet3 Training loss and training accuracy VS epochs

The graph depicts the increase in training accuracy and a corresponding decrease in training loss with the increasing number of epochs fully convolutional neural network with layers. The training accuracy obtained at the 800th epoch is around 0.965 percent and the lowest loss that was attained is around 0.12 percent.

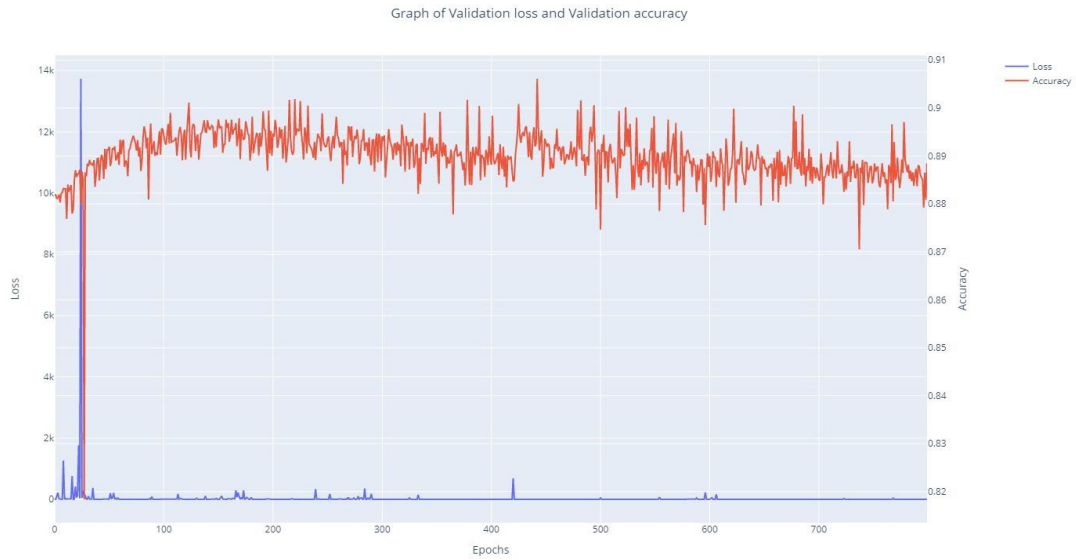


Figure 16: Unet 3 – Validation loss and validation accuracy VS epochs

The highest validation accuracy obtained at the 800th epoch is around 0.908 percent and the corresponding validation loss that was attained is around 0.001 percent.

After analyzing minutely, the graphs of validation loss, validation accuracy, training loss, and training accuracy. Unet 3 seems to attain the highest accuracy out of the others. Fully convolutional models also give a pretty convincing result at a tradeoff of overfitting the data after 500 iterations. At some points, the accuracy line drops low which indicates that the model is underfitting or possibly not able to learn further which is taken care of by the Adam optimizer which introduces the learning rate and allows the model to overcome the local minima and increase the accuracy after some time.

5.3 Network Outcome and Accuracy Assessments:

I. Representation and Assessment of Fully Connected Network (5 layers) on 4 different patches:

a. Visual Representation and Accuracy assessment of Flevoland 107:

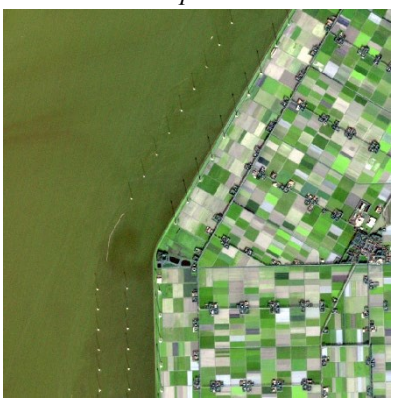


Figure 17-a: Flevoland 107 Original

Figure 17-b: Flevoland 107 GT

Figure 17-c: Flevoland 107 Prediction

Confusion Matrix: flevoland 107

	Actual Other	Actual Field Boundary	Sum
Prediction Other	595095 (92.984%)	16646 (2.601%)	611741
Prediction Field Boundary	11087 (1.732%)	17172 (2.683%)	28259
Sum	606182	33818	

Overall Accuracy	95.667	%
Precision	60.766	%
Recall	50.778	%
F1 Score	55.325	%

Table 2: Flevoland 107 accuracy assessments

b. Visual Representation and Accuracy assessment of Friesland 177:

Figure 18-a: Friesland 177 Original

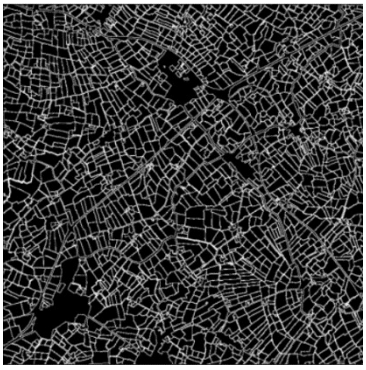


Figure 18-b: Friesland 177 GT

Figure 18-c: Friesland 177 Prediction

Confusion Matrix: friesland 177

	Actual Other	Actual Field Boundary	Sum
Prediction Other	473114 (73.924%)	58429 (9.13%)	531543
Prediction Field Boundary	30112 (4.705%)	78345 (12.241%)	108457
Sum	503226	136774	

Overall Accuracy	86.165	%
Precision	72.236	%
Recall	57.281	%
F1 Score	63.895	%

Table 3: Friesland 177 accuracy assessments

c. Visual Representation and Accuracy assessment of Overjissel 4:

Figure 19-a: Overjissel 4 Original

Figure 19-b: Overjissel 4 GT

Figure 19-c: Overjissel 4 Prediction

Confusion Matrix: overijssel 4

	Actual Other	Actual Field Boundary	Sum
Prediction Other	504646 (78.851%)	65111 (10.174%)	569757
Prediction Field Boundary	24912 (3.892%)	45331 (7.083%)	70243
Sum	529558	110442	

Overall Accuracy	85.934	%
Precision	64.535	%
Recall	41.045	%
F1 Score	50.177	%

Table 4: Overijssel 4 accuracy assessments

Visual Representation and Accuracy assessment of Overjissel 31:

Figure 20-a: Overjissel 31 Original

Figure 20-b: Overjissel 31 GT

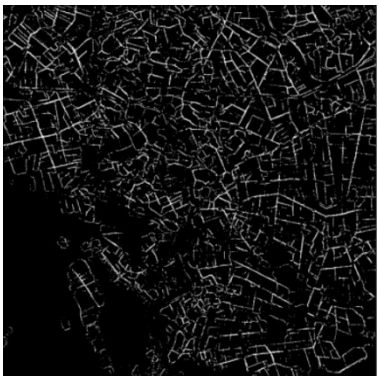


Figure 20-c: Overjissel 31 Prediction

Confusion Matrix: overijssel 31

	Actual Other	Actual Field Boundary	Sum
Prediction Other	519636 (81.193%)	70004 (10.938%)	589640
Prediction Field Boundary	16319 (2.55%)	34041 (5.319%)	50360
Sum	535955	104045	

Overall Accuracy	86.512	%
Precision	67.595	%
Recall	32.718	%
F1 Score	44.093	%

Table 5: Overijssel 31 accuracy assessments

II. Representation and Assessment of Fully Connected Network (6 layers) on 4 different patches:

a. Visual Representation and Accuracy assessment of Flevoland 80:

Figure 21-a: Flevoland 80 Original

Figure 21-b: Flevoland 80 GT

Figure 21-c: Flevoland 80 Prediction

Confusion Matrix: flevoland 80

	Actual Other	Actual Field Boundary	Sum
Prediction Other	584522 (91.332%)	15354 (2.399%)	599876
Prediction Field Boundary	15787 (2.467%)	24337 (3.803%)	40124
Sum	600309	39691	

Overall Accuracy	95.134	%
Precision	60.654	%
Recall	61.316	%
F1 Score	60.984	%

Table 6: Flevoland accuracy assessments

b. Visual Representation and Accuracy assessment of Friesland 165:

Figure 22-a: Friesland 165 Original

Figure 22-b: Friesland 165 GT

Figure 22-c: Friesland 165 Prediction

Confusion Matrix: flevoland 80

	Actual Other	Actual Field Boundary	Sum
Prediction Other	584522 (91.332%)	15354 (2.399%)	599876
Prediction Field Boundary	15787 (2.467%)	24337 (3.803%)	40124
Sum	600309	39691	

Overall Accuracy	95.134	%
Precision	60.654	%
Recall	61.316	%
F1 Score	60.984	%

Table 7: Flevoland 80 accuracy assessments

c. Visual Representation and Accuracy assessment of Overjissel 58:

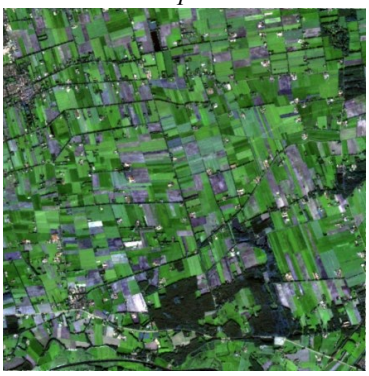


Figure 23-a: Overjissel 58 Original

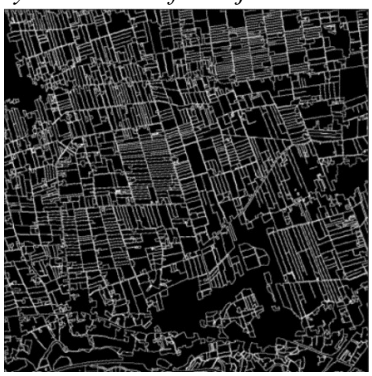


Figure 23-b: Overjissel 58 GT

Figure 23-c: Overjissel 58 Prediction

Confusion Matrix: overijssel 58

	Actual Other	Actual Field Boundary	Sum
Prediction Other	490643 (76.663%)	57187 (8.935%)	547830
Prediction Field Boundary	33596 (5.249%)	58574 (9.152%)	92170
Sum	524239	115761	

Overall Accuracy	85.815	%
Precision	63.55	%
Recall	50.599	%
F1 Score	56.34	%

Table 8: Overijssel 58 accuracy assessments

d. Visual Representation and Accuracy assessment of Zeeland 19:

Figure 24-a: Zeeland 19 Original

Figure 24-b: Zeeland 19 GT

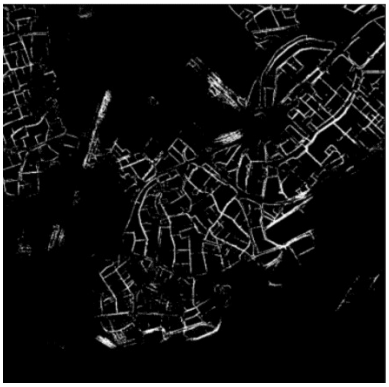


Figure 24-c: Zeeland 19 Prediction

Confusion Matrix: zeeland 19

	Actual Other	Actual Field Boundary	Sum
Prediction Other	573209 (89.564%)	34694 (5.421%)	607903
Prediction Field Boundary	14707 (2.298%)	17390 (2.717%)	32097
Sum	587916	52084	

Overall Accuracy	92.281	%
Precision	54.18	%
Recall	33.388	%
F1 Score	41.316	%

Table 9: Zeeland 19 accuracy assessments

III. Representation and Assessment of Unet 3 on 4 different patches:

a. *Visual Representation and Accuracy assessment of Friesland 164:*

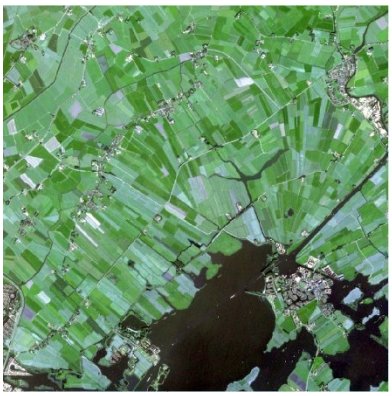


Figure 25-a: Friesland 164 Original

Figure 25-b: Friesland 164 GT

Figure 25-c: Friesland 164 Prediction

Confusion Matrix: friesland 164

	Actual Other	Actual Field Boundary	Sum
Prediction Other	494174 (77.215%)	34878 (5.45%)	529052
Prediction Field Boundary	30786 (4.81%)	80162 (12.525%)	110948
Sum	524960	115040	

Overall Accuracy	89.74	%
Precision	72.252	%
Recall	69.682	%
F1 Score	70.944	%

Table 10: Friesland 164 accuracy assessments

b. Visual Representation and Accuracy assessment of Friesland 166:

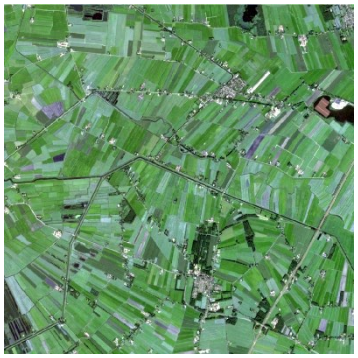


Figure 26-a: Friesland 166 Original

Figure 26-b: Friesland 166 GT

Figure 26-c: Friesland 166 Prediction

Confusion Matrix: friesland 166

	Actual Other	Actual Field Boundary	Sum
Prediction Other	447372 (69.902%)	52264 (8.166%)	499636
Prediction Field Boundary	41037 (6.412%)	99327 (15.52%)	140364
Sum	488409	151591	

Overall Accuracy	85.422	%
Precision	70.764	%
Recall	65.523	%
F1 Score	68.043	%

Table 11: Friesland 166 accuracy assessments

c. Visual Representation and Accuracy assessment of Overjissel 20:

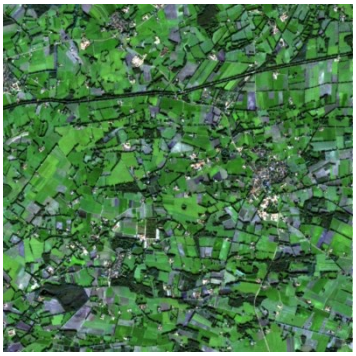


Figure 27-a: Overjissel 20 Original

Figure 27-b: Overjissel 20 GT

Figure 27-c: Overjissel 20 Prediction

Confusion Matrix: overijssel 20

	Actual Other	Actual Field Boundary	Sum
Prediction Other	485747 (75.898%)	44707 (6.985%)	530454
Prediction Field Boundary	39440 (6.162%)	70106 (10.954%)	109546
Sum	525187	114813	

Overall Accuracy	86.852	%
Precision	63.997	%
Recall	61.061	%
F1 Score	62.494	%

Table 12: Overijssel 20 accuracy assessments

a. Visual Representation and Accuracy assessment of Zeeland 32:

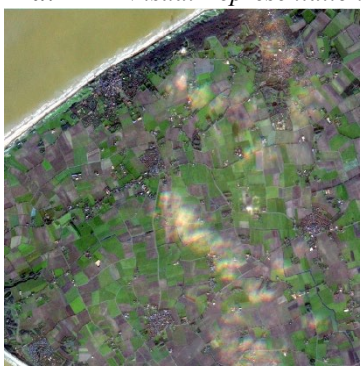


Figure 28-a: Zeeland 32 Original

Figure 28-b: Zeeland 32 GT

Figure 28-c: Zeeland 32 Prediction

Confusion Matrix: zeeland 32

	Actual Other	Actual Field Boundary	Sum
Prediction Other	532454 (83.196%)	31397 (4.906%)	563851
Prediction Field Boundary	26817 (4.19%)	49332 (7.708%)	76149
Sum	559271	80729	

Overall Accuracy	90.904	%
Precision	64.784	%
Recall	61.108	%
F1 Score	62.892	%

Table 13: Zeeland 32 accuracy assessments

5.4 Comparison and Analysis of Networks:

After a detailed analysis of predictions made by different networks, it is observed that Unet 3 performs best for both sparsely distributed fields and densely distributed fields and gives the optimal visual prediction for almost all the test images. A fully connected network with 5 and 6 layers gives somewhat good predictions for sparely distributed fields but Unet 3 which was originally designed for the image segmentation of biomedical images performs the best in every scenario.

It was expected that images with 2 to 3 percent cloud cover would pose a challenge while predicting but surprisingly it gave clear boundaries even in these scenarios. The dynamic ability of the networks could be attributed to the fact that clear predictions were made even for those patches where the fields were not in a pattern. The neural networks, especially Unet 3 were able to distinguish between boundaries in the fielded areas and boundaries in city areas. Also, coastal areas were in minority in the overall dataset so it was thought that the network would find it difficult to detect the field boundaries along the coastal areas but on the contrary, it was found that field boundaries near the coastal line were precisely detected and delineated. In some images, the contrast between some fields is not that clear which posed a little challenge for prediction.

6. Discussions:

In this section, the overall result intelligence has coalesced and further steps are required to process the output delineated fields for overlaying on the original eo patches.

6.1 Choice of data and network:

As far as the problem of field delineation is concerned, the quality, quantity, and temporal complexity of the earth observation images are crucial for training the data. If the images are not adequate for a particular area, then data from another area can be chosen and implemented on the AOI for predictions, data augmentation techniques can be used for increasing the number of images, and data masking techniques can be used for improving the image quality. The high resolution of EO images also makes a huge difference in the predictions made by the network.

6.2 Evaluation metrics:

a. Precision:

Precision is computed by dividing the amount of relevant positive predictions by the number of genuine positives. Real-world models rarely attain 100% accuracy and 100 percentage recall, but a perfect machine learning classifier model may. Prediction accuracy is inevitably a trade-off in modeling. Usually, the recall is lesser the greater the precision, and vice-versa. It mainly tries to answer the question "How many of the affirmative identifications were truly accurate?" It is given as:

$$\text{Precision} = \frac{TP}{TP + FP}$$

In the network's accuracy assessment, we get an average precision of about 65 percent which indicates that the model can classify the correct boundary pixels.

b. Recall:

Recall, in contrast to Precision, is unaffected by the quantity of incorrect sample categorization. Additionally, Recall will be 1 if the system labels all positive data as affirmative. The question in highlight for the recall is "What percentage of true positives were detected?". The models can identify the actual positives or the true positives correctly around 60 percent of the time. The recall is given as:

$$\text{Recall} = \frac{TP}{TP + FN}$$

c. F1 Score:

The harmonic average of recall and precision is what the F1-score is by description. Because it charges uneven values more severely, the F1 score employs a harmonic mean. As this score takes both false positives and false negatives into account. It is preferable to include both Precision and Recall if the costs of false positives and false negatives are distinguishable. We get an F1 score of about 62 percent in our different network analyses which translates into good boundary classification. F1 score is given as:

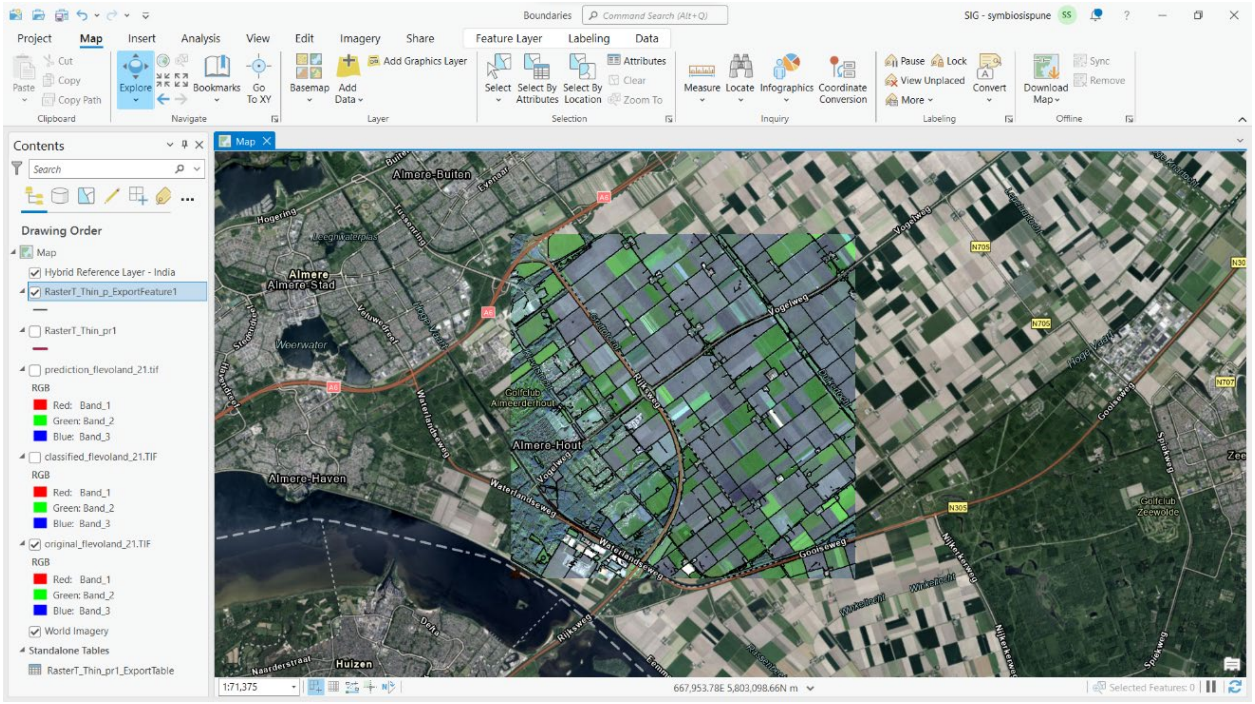
$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

6.3 Postprocessing:

The output predictions are in the form of tiff images which are sufficient if it is just to evaluate the performance of the network but for geospatial applications, these images need to be georeferenced to their original layer and the boundaries need to be vectorized i.e., the output should be in the form of polygon shapefiles which can be integrated on field maps or field satellite imagery for precision farming.

To achieve that end of the problem, ArcGIS software was used to post-process the images. Network output tiff images were (1) converted from raster to vector (2) georeferenced then (3) polygonized into shapefile. Some unnecessary polygons which arise due to pixel delocalization were dissolved to the most adjacent polygon.

An example can be seen in the figure below, which shows the shapefile which is created after the postprocessing of the predicted tiff image given by the Unet 3 network.



7. Conclusion:

The obvious result that has surfaced out of this project is that UNet 3 performs the best for this purpose but fully convolutions are also a viable choice. For future works, training the networks with a larger amount of data from different countries with some data augmentation on a higher number of epochs can be possible with some funding. This will fill in the room for improvement and obtain absolute ground truth-like results.

A lot of time can be given to automating and developing the process of converting the outputted tiff images from neural networks to polygon files. The task of field delineation is challenging and this work did its best in the available time and resources to overcome that challenge.

8. References:

Journal Articles:

1) Franc¸ois Waldnera, Foivos Diakogiannisb , “Deep learning on edge: extracting field boundaries from satellite images with a convolutional neural network”, Journal: *CSIRO Agriculture and food,* 2019.

2) Tanmay Anand, Soumendu Sinha, Murari Mandal, Vinay Chamola, and F. Richard Yu, “AgriSegNet: Deep Aerial Semantic Segmentation Framework for IoT-assisted Precision Agriculture”, Journal: *IEEE*, 2021.

3) FrançoisWaldner, Foivos I. Diakogiannis, Kathryn Batchelor, Michael Ciccotosto-Camp, Elizabeth Cooper-Williams, Chris Herrmann, Gonzalo Mata, and Andrew Toovey, “Scalable Mapping of Field Boundaries Using Satellite Images”, Journal: *Remote Sensing Journal,* 2021.

4) Ruoyu Yang, Zia U. Ahmed, C. Schulthess, Mustafa Kamal, Rahul Rai, “Detecting functional field units from satellite images in smallholder farming systems using a deep learning based computer vision approach: A case study from Bangladesh”, Journal: *Remote Sensing Applications,* 2020.

5) Iasonas Kokkinos, “Pushing the boundaries of boundary detection using deep learning”, Journal: *ICLR,* 2016.

6) Rishi Kumar Suresh Kumar, “Automated Field Boundary Detection Using Modern Machine Learning Techniques”, Journal: Journal of Computer Engineering, 2020.

7) Saining Xie, Zhuowen Tu, “Holistically-Nested Edge Detection”, Journal: *Computer Vision and Pattern Recognition*, 2015.

8) L. Meyer, F. Lemarchand, P. Sidiropoulos, “A deep learning architecture for batch-mode fully automated field boundary detection”, Journal: *Remote Sensing and Spatial Information Sciences*, 2020.

9) Surabhi Lingwal, Komal Kumar Bhatia and Manjeet Singh, “Semantic segmentation of landcover for cropland mapping and area estimation using Machine Learning techniques”, Journal: *Data Intelligence*, 2022.

10) Alireza Taravat, Matthias P. Wagner, Rogerio Bonifacio, and David Petit, "Advanced Convolutional Networks for Agricultural Field Boundary Detection", Journal: *Remote Sensing Journal*, 2021.

Online Articles:

1) Dice loss for crisp boundaries
2) Image Segmentation model – Unet
3) Unet for biomedical Image Segmentation
4) Logits in Deep learning
5) Softmax Activation Function
6) Understanding Fully Convolutional Network
7) Understanding different activation functions in Neural networks
8) Georeference a raster to another raster

9. Appendix: (Source Code)

a. Data Preprocessing:

The following code was written for the preprocessing of GeoTIFF images. The images were first normalized using NDVI then sampled and divided into smaller patches.

```
def normalize(val,min,max):
  return (val-min)/(max-min)

# Using the formula of ndvi we define the normalize function
# normalize_array function which will take the 3d arrays as input and iterate over bands and normalize them one by o

def normalize_array(arr):

  # np.zeros will set array with zeros in the form of image shape (800*800)
  arr_norm = np.zeros(arr.shape,dtype=np.float32)

  # The for loop will take in range all the bands in the images and compute the minimum and the maximum of
  # those image bands and store it in 'arr_norm' as normalized pixel values.
  for i in range(arr.shape[2]):
    min = arr[:,:,i].min()
    max = arr[:,:,i].max()
    arr_norm = (arr - min)/(max - min)
  return arr_norm

# k will take all the images from images dictionary and noramlize it accordingly by taking the same image through v.
for k,v in images.items():
  images[k] = normalize_array(v)
  print(f"Normalized {k[0]}_{k[1]}")
```

```
def gridwise_sampling(img,patchsize):
  x = int(img.shape[0] / patchsize)
  y = int(img.shape[1] / patchsize)
  n = img.shape[2]
  s = patchsize

  patches = np.zeros(shape = (0,s,s,n), dtype = img.dtype)

  for row in range(x):
    for col in range(y):
      patch = img[row * s : (row + 1) * s, col * s : (col + 1) * s, :]
      patch = np.expand_dims(patch, axis=0)
      patches = np.concatenate((patches,patch),axis = 0)
  return patches

x_train = np.zeros(shape = (0,PATCHSIZE, PATCHSIZE, NBANDS),dtype = np.float32)
y_train = np.zeros(shape = (0, PATCHSIZE,PATCHSIZE,1),dtype = np.int)

for k in x_train_dict.keys():
  x_patches = gridwise_sampling(x_train_dict[k], PATCHSIZE)
  y_patches = gridwise_sampling(y_train_dict[k], PATCHSIZE)

  x_train = np.concatenate((x_train, x_patches), axis = 0)
  y_train = np.concatenate((y_train,y_patches), axis =0)

sample_size = 10

while sample_size:
  visualize_data(x_train[sample_size], f"Patch sample {sample_size}")
  visualize_labels(y_train[sample_size], f"Patch sample {sample_size}", LEGEND)
  sample_size -= 1
```

For splitting the dataset, the following lines of codes were written,

```
import random
seed = 34
relative_train_size = 0.9

def calculate_k(n):
  k = relative_train_size *n
  return int(k)

# We will create a random generator A funtion to calculate the k
random_gen = random.Random(seed)

# We will extract all the keys from the image dictionary
keys = set(images.keys())

# Then we will split the keys set into training and testing keys
# The keys_train' variable stores the random sample from the images dictionary by using the sample(sequence, sampling_size) function,
# in our case the sequence is keys from images dict and sampling size is 90 percent of images.

keys_train = set(random_gen.sample(keys,calculate_k(len(keys))))
keys_test = keys - keys_train


# Now the y_train variable will also have the same keys in labels as in images so it will store the corresponding classification labels
# If k is 0 then it will pickup the corresponding value from keys_train and the same value will be chosen from images dict as the province name is same
# and it will be inputed into the x_train_dict as key and its corresponding value will be stored.

x_train_dict = {k : images[k] for k in keys_train}
y_train_dict = {k : labels[k] for k in keys_train}

# k in the top is different it will give the percent of split wheres k in the bottom part is just an iterator
x_test_dict = {k : images[k] for k in keys_test}
y_test_dict = {k : labels[k] for k in keys_test}
```

b. For setting up a neural network:

Tensorflow's Keras library was used for setting and building the neural network through the following lines of codes:

```
from keras.layers import Activation, BatchNormalization, Convolution2D, LeakyReLU, Reshape, ZeroPadding2D
from keras.models import Sequential
from tensorflow.keras.optimizers import SGD, Adam
from keras_unet.models import satellite_unet


def build_unet(x: int, y: int, bands: int, labels: int, layers: int =2,) -> tf.keras.Model:
  model = satellite_unet(input_shape = (x,y,bands),
                         num_classes = labels,
                         output_activation = "softmax",
                         num_layers = layers)
  return model


def build_unet2(x:int, y:int, bands: int, labels:int,) -> tf.keras.Model:
  return build_unet(x, y, bands, labels, layers = 2)


def build_unet3(x:int, y:int, bands:int, labels:int,):
  return build_unet(x,y,bands,labels, layers = 3)

def build_fcndk(x:int, y:int, bands:int, labels: int, layers = 4,) -> tf.keras.Model:
  model = keras.models.Sequential()
  model.add(ZeroPadding2D((2,2), input_shape = (x,y,bands)))
  model.add(Convolution2D(filters = 16,
                          kernel_size = (5,5),
                          dilation_rate = (1,1)))
  model.add(BatchNormalization(axis = 3))
  model.add(LeakyReLU(0.1))
```

```
  if layers >= 5:
    # FCNDK5
    model.add(ZeroPadding2D((10, 10)))
    model.add(Convolution2D( filters=32,
                             kernel_size=(5, 5),
                             dilation_rate=(5, 5)))

    model.add(BatchNormalization(axis=3))
    model.add(LeakyReLU(0.1))

  if layers >= 6:
    # FCNDK6
    model.add(ZeroPadding2D((12, 12)))
    model.add(Convolution2D( filters=32,
                             kernel_size=(5, 5),
                             dilation_rate=(6, 6)))

    model.add(BatchNormalization(axis=3))
    model.add(LeakyReLU(0.1))

  # output layer
  model.add(Convolution2D (filters = labels,
                           kernel_size = (1,1)))
  model.add(keras.layers.Activation(activation = "softmax"))
  return model
```

```
def build_fcndk3(x:int, y:int, bands: int, labels: int,) -> tf.keras.Model:
  return build_fcndk(x,y,bands,labels, layers = 3)

def build_fcndk4(x:int, y:int, bands: int, labels: int,) -> tf.keras.Model:
  return build_fcndk(x,y,bands,labels, layers = 4)

def build_fcndk5(x:int, y:int, bands: int, labels: int,) -> tf.keras.Model:
  return build_fcndk(x,y,bands,labels, layers = 5)

def build_fcndk6(x:int, y:int, bands: int, labels: int,) -> tf.keras.Model:
  return build_fcndk(x,y,bands,labels, layers = 6)


def build_network(name:str)-> typing.Callable:
  if name.lower() == "fcndk3":
    return build_fcndk3
  elif name.lower() == "fcndk4":
    return build_fcndk4
  elif name.lower() == "fcndk5":
    return build_fcndk5
  elif name.lower() == "fcndk6":
    return build_fcndk6
  elif name.lower() == "unet2":
    return build_unet2
  elif name.lower() == "unet3":
    return build_unet3
```

c. Training the Neural Network:

The model was trained by executing this code block:

```
if exec_mode in [TRAIN_NEW]:
  print(f"Built a new network: {NETWORK_UUID} {NETWORK_NAME}")
  model = model_builder(x_train.shape[1], x_train.shape[2], NUMBER_BANDS, NUMBER_CLASSES)
  model.compile(optimizer = OPTIMIZER, loss = 'binary_crossentropy',
                metrics = 'accuracy')

if exec_mode in [LOAD, LOAD_AND_TRAIN]:
  print(f"Loaded the existing network: {NETWORK_UUID} {NETWORK_NAME}")
  m = import_model(NETWORK_UUID, NETWORK_NAME)
  readme = m.readme
  model = m.model
  history = m.history

print(readme)

if exec_mode in [TRAIN_NEW, LOAD_AND_TRAIN]:
  y_train = to_categorical_4d(y_train, NUMBER_CLASSES)
  history = train(model, x_train, y_train, NUMBER_EPOCHS,
                  batch_size = BATCH_SIZE,
                  validation_split = VALIDATION_SPLIT).history

if exec_mode in [TRAIN_NEW, LOAD, LOAD_AND_TRAIN]:
  if SAVE_AFTER_EXEC:
    export_model(ModelHist(NETWORK_UUID, NETWORK_NAME, model, history, readme))

if exec_mode in [SKIP]:
  print("Remark: Execution mode is chosen as skip, Nothing will be executed")
```

d. For visualizing the network performance:

```
from plotly.subplots import make_subplots

def generate_plot(metric1, metric2, title: str):
  if title is None:
    title = p

  # Create figure with secondary y-axis
  fig1 = make_subplots(specs=[[{"secondary_y": True}]])
  # Add traces
  fig1.add_trace(go.Scatter(y=metric1, name="Loss"),
                 secondary_y=False)

  fig1.add_trace(go.Scatter(y=metric2, name="Accuracy"),
                 secondary_y=True)

  fig1.update_layout(title=f"Graph of {title} loss and {title} accuracy",title_x=0.50)
  fig1.update_xaxes(title_text="Epochs")
  fig1.update_yaxes(title_text="Loss", secondary_y = False)
  fig1.update_yaxes(title_text="Accuracy", secondary_y = True)
  fig1.update(layout_coloraxis_showscale = True)
  fig1.update_layout(height = 800)
  fig1.update_layout(hovermode="x unified")

  fig1.show()

generate_plot(history["loss"],history["accuracy"],"Training")
generate_plot(history["val_loss"], history["val_accuracy"], "Validation")
```

e. To visualize the predictions:

The following technique was employed for predicting the field boundaries

```
def print_map(labels: np.ndarray, title:str):
    """Will print output based on given input labels.

    Coloring:
        Label 1 = other = black
        Label 2 = boundary = white

    Remark: The expected data structure is a 2 dimensional array with the int
            value of the corresponding label.

    :param labels: The 2D array of labels per pixel
    :param title: The plot title
    """
    # creating the image in RGB

    x,y = labels.shape[0], labels.shape[1]
    img = np.zeros((x,y,3), dtype = np.uint8)

    for i in range(img.shape[2]):
      img[:,:,i] = np.where(labels[:,:] == 2,255,0)

    # drawing the plot
    fig = pyplot.figure(figsize = (7,7))
    pyplot.imshow(img)
    pyplot.suptitle(title)
    pyplot.show()
```

f. To get the accuracy assessment of predictions:

The code for running an accuracy assessment on the predictions is given below:

```
def print_binary_confusion_matrix(title, tp, fp, fn, tn):
    cm = [tp, fp, fn, tn]
    total = sum(cm)
    # percentages
    #(ptp, pfp, pfn, ptn) = tuple([int(100 * cm[i]) for i in range(len(cm))])
    (ptp, pfp, pfn, ptn)= tuple([100 * cm[i] / total for i in range(len(cm))])

    header_row = ["Actual Other", "Actual Field Boundary", "Sum"]
    header_col = ["Prediction Other", "Prediction Field Boundary", "Sum"]

    data = [
        [f"{tn} ({round(ptn, 3)}%)", f"{fn} ({round(pfn, 3)}%)", tn + fn],
        [f"{fp} ({round(pfp, 3)}%)", f"{tp} ({round(ptp, 3)}%)", fp + tp],
        [tn + fp, fn + tp, ""]
    ]

    df = pandas.DataFrame(data, header_col, header_row)
    print(f"Confusion Matrix: {title}")
    print("=" * 32)
    display(df)
    print("\n" * 3)
```

g. For exporting the Predictions:

The predictions made by the neural networks are exported to the specified path using these lines of code:

```
for f, keys in keys_per_province.items():
    for k in keys:
        x = x_dict[k]

        (_, _, f_weights, _) = get_file_names(NETWORK_UUID, NETWORK_NAME)

        try:
            img = evaluate_predictions(
                x,
                NUMBER_CLASSES,
                f_weights,
                OPTIMIZER,
                model_builder,
            )
        except Exception as e:
            print(f"Failed to predict image of size {x.shape[0]}x{x.shape[1]}")
            continue

        # Build directory name
        fname = DATA_PATH
        fname += f"{f}/"
        fname += f"Prediction_{NETWORK_NAME}/"

        # Create directory
        pathlib.Path(fname).mkdir(parents=True, exist_ok=True)

        # build filename
        fname += f"prediction_{k[0]}_{k[1]}"
        fname += ".tif"

        export_array(img, fname)
        print(f"Exported prediction {fname}")
```

Turnitin Originality Report

Processed on: 24-Aug-2022 13:50 IST
ID: 1886321368
Word Count: 5672
Submitted: 1

Arbaz Sayed Project report By Arbaz Sayed

Similarity Index	Similarity by Source	
2%	Internet Sources:	1%
	Publications:	2%
	Student Papers:	1%

1% match (Internet from 04-Sep-2021)
https://www.sciencegate.app/keyword/559289

1% match (publications)
François Waldner, Foivos I. Diakogiannis, Kathryn Batchelor, Michael Ciccotosto-Camp et al. "Detect, Consolidate, Delineate: Scalable Mapping of Field Boundaries Using Satellite Images", Remote Sensing, 2021

< 1% match (student papers from 26-Jul-2022)
Submitted to College of Engineering & Technology Bhubaneswar on 2022-07-26

< 1% match (student papers from 28-Feb-2022)
Submitted to Dartford Grammer School on 2022-02-28

< 1% match (Internet from 17-Jun-2021)
https://repository.cardiffmet.ac.uk/bitstream/handle/10369/3020/Neal%20Brooker.docx?isAllowed=y&sequence=1

< 1% match (Internet from 02-May-2022)
https://www.hindawi.com/journals/cin/2022/4567989/

< 1% match (Internet from 18-Oct-2021)
https://uwspace.uwaterloo.ca/bitstream/handle/10012/17430/Ji_Zongliang.pdf?isAllowed=y&sequence=3

CHAPTER - I INTRODUCTION Boundary detection for Field Delineation from GeoTIFF Images Using Neural Networks 1. Introduction An important step in various agriculture-related remotely sensed pipelines is the precise segmentation of huge amounts of land. Using a state-of-the-art deep learning algorithm with nothing more than a singular Sentinel-2 picture as feed, this research attempts to entirely simplify this time-consuming and resource-intensive operation. Identifying the boundaries that separate one cropland from the other on the field has been one of the most challenging tasks due to many obstacles such as cloud cover, the gap between machine learning techniques and satellite images, and spatial-temporal limitations of satellite data. GeoTIFF images are raster image files that store the satellite or drone mapped images and retain their quality even after they are compressed or manipulated which is ideal for working with imagery data for this matter. For bridging the gap between earth observation data and the current machine and deep learning techniques EO learn (Earth Observation) library was