School of GeoSciences

Dissertation

for the degree of

MSc in Geographical Information Science

Anja Helfrich

August 2020

THE UNIVERSITY of EDINBURGH

Detection of Roof Type in Rural Tanzania using High-Resolution Satellite Imagery and Convolutional Neural Networks

Part I: Research Paper

Statements of Originality and Copyright

I declare that this dissertation represents my own work, and that where the work of others has been used it has been duly accredited. I further declare that the length of components for this dissertation are **5152** words for the Research Paper and **2520** words for the Technical Report.

I agree that this dissertation and associated electronic documents, web pages, data, files and computer programs can be retained by the University. YES

I agree that, with the permission of my supervisor(s) or the Programme Director, these materials be made available for the purposes of preparing a publication. YES

I agree that, with the permission of my supervisor(s) or the Programme Director, these materials can be used within the University of Edinburgh for continued research. YES

Signature: Anja Helfrich *Date: 13th August 2020*

Acknowledgements

I'd like to thank my supervisor, Gary Watmough for and his support and willingness to let me run wild in my ideas on how to accomplish the task at hand, and for his suggestions on analysis and how to tie up the loose ends of the project. I'd like to thank Isaac Neal and Dr. Seth Sohan from the Informatics department for their support and expert advice in machine learning. I'd also like to thank Chris Hill for his IT support and encouragement in helping me start the process of setting up my remote technical environment. Finally, I'd like to thank my partner Samuel Abbanat for supporting me throughout the double challenges of COVID-19 and completing this dissertation.

ABSTRACT

The United Nations (UN) efforts to completely eliminate poverty by 2030 have been less successful in rural areas of Sub Saharan Africa compared to other parts of the world. This is partially due to infrequent and incomplete collection of census and other poverty-predictive data necessary to monitor progress and target resources to the communities most affected by poverty. The recent increase in available satellite data and development of machine learning (ML) tools could provide a way to increase accuracy and frequency of data collection for the improved monitoring of socioeconomic change. This study aims to use Convolutional Neural Networks (CNNs), a type of ML, to automatically extract roof type information from high-resolution satellite imagery of Mbola, Tanzania as changes in roof material are thought to be a spatially-predictive indicator of socioeconomic change. While the model reaches an accuracy of 85% for the detection of metal roofs in Mbola, this accuracy drops when the model is applied to different countries and time-periods, and thatch roofs are significantly more difficult to detect. This study highlights the need for further development of ML analysis techniques to better understand patterns extracted from data and to improve the efficiency of the training process. The development of more ML training data specific to African Earth Observation (EO) tasks is key before CNNs become a consistently viable tool for the monitoring of poverty in African countries.

Part I: Research Paper

1. Introduction

1.1. Socioeconomic data collection in Sub-Sahara Africa

Estimating current populations and socioeconomic levels is key in targeting resources to areas of socioeconomic distress to achieve the goals set by local and global organizations dedicated to combating poverty (Engstrom et al., 2017). While this information has previously been collected through census and household surveys, these methods are expensive and therefore infrequent (World Bank, 2015). The United Nations (UN) Sustainable Development Goals (SGDs) include seventeen goals ranging from providing food and job security to ending gender discrimination and providing education (UN, 2019). The goal to completely eliminate poverty by 2030 has been largely successful in many areas of the world, with poverty decreased by more than half in Eastern Asia within the last decade. However, significantly less improvement has been made in Sub-Saharan Africa, which accounted for 56% of those living under the poverty line in 2015, and this gap in improvement continues to increase (UN, 2019). The lack of improvement in Sub-Saharan African countries is exacerbated by the lack of data and spatial data infrastructures available for these areas, especially in sparsely populated, rural communities which are often associated with the highest levels of poverty (Guigoz et al., 2017) (IMF, 2011) (IFAD, 2019).

With the recent onset and continuing effects of the global COVID-19 pandemic, an estimated 70 million people (23 million of which are in Sub-Saharan Africa) will soon fall under the poverty line, indicating the first increase in global poverty in decades. This setback has given birth to the United Nations COVID-19 Response and Recovery Fund, which aims to alleviate these devastating effects (UN, 2020). The impact of COVID-19 and the measures that are being taken to counteract it highlight the need more than ever for accurate, up-to-date information to target resources more efficiently to communities in need.

1.2. Roof type as a socioeconomic indicator

The need for additional sources of data stems from more than just a lack of census data. Income and other census-derived indicators of poverty alone are not always spatially reliable predictors of poverty (Okwi et al., 2017). There are several indicators of socioeconomic status which can be extracted from satellite data and used to predict aspects of poverty. A study conducted in India found that elevation, distance to urban areas and nighttime lights were the most predictive factors of rural poverty (Steele et al., 2017). Another study in Kenya found indicators of rural poverty to include size of buildings within a homestead, bare ground surrounding a homestead, and length of growing season. This study also found that the use of satellite data could predict the poorest households in Kenya with an accuracy of 62% (Watmough et al., 2019). A study of five developing countries in Africa found roofing material to have an almost linear relationship with indicators of poverty such as household expenditures (Jean et al., 2016). Roof material was further supported as a robust indicator of spatial variation in poverty in a study of high-resolution satellite features correlated with measures of well-being in Sri-Lanka (Engstrom et al., 2017).

With the ever-increasing store of higher-resolution and more temporally frequent satellite data, mapping changes in roof type could prove to be a cost-effective supplement to the lack of socioeconomic data for rural communities in Africa. Further automated extraction of this data is needed to reach the goal of improved SGD monitoring.

1.3. Methods for detecting roof material type

Previous studies have shown that object-oriented software such as Ecognition (Trimble Inc., 2019) can be used to develop workflows to detect different roof types in specific areas of rural Africa with an accuracy of up to 85% using high-resolution satellite imagery (Campbell, 2019). However, when testing these workflows on communities in different countries, the accuracy drops significantly due to the different classification characteristics of roofs in different areas.

Using machine learning (ML) algorithms in place of software-based workflows, in which rigid parameters must be set based on location, could increase the accuracy of roof detection/classification as well as the flexibility of such a model to be used on different landscapes. This is because ML algorithms, as non-parametric classifiers, don't assume that a class of data (such as roofs) has a normal distribution. This allows for more flexibility in the model by allowing the model to 'learn' a more loose or diverse definition of what a roof looks like in different settings (Song et al., 2019).

Convolutional Neural Networks (CNNs) are a type of deep ML that have been shown to produce overall higher accuracies compared to other ML techniques such as Support Vector Machine (SVM) techniques and Random Forest models when it comes to object detection (Miyamoto et al., 2018). CNN's are especially good for complex data, such as the complexity inherent in multispectral remote sensing images. This is because of their ability to use a network of connected data layers that contain modifiable weights per pixel to backpropagate accuracy information and adjust those weights to perform calculations that will produce a more accurate outcome (Ruscica, 2020; also See Technical Report Section 3).

The three main types of ML object-oriented tasks are classification, localization, and object detection. Classification is labeling an image as one class. Localization is identifying the location of one object in an image. Object detection is locating and classifying multiple objects in an image. Classification and localization can be thought of as the broken down aspects that make up the more complicated goal of object detection (Brownlee, 2019). Segmentation is also possible using CNNs but is often the means to an end involving object identification and does not utilize spatially-relative spectral and textural information (Längkvist et al., 2016; Jozdani et al., 2019).

There are two broad approaches for developing a CNN. The first involves repurposing a successfully trained model for the task at hand. This approach is called 'transfer learning' and has been widely successful on datasets for which there is an abundance of labeled training data (Salcedo-Sanz et al., 2020). However, if training data is limited there is often a risk of 'over-parametrization' or 'over-fitting', which involves an excess of adjustable weights for the amount of data available and can cause erratic training. This is because well-established CNN architectures are often designed specifically for large quantities of computer vision data, and subsequently have complex structures, which produce many trainable parameters. With limited training data, there is a need to reduce parameters so that the model converges smoothly in the training process. This requires a simple architecture with fewer convolutional layers (Li et al., 2019; Song et al., 2019).

1.4. Challenges in Object Detection specific to Earth Observation

The development of ML models has skyrocketed in the last few years to meet the data processing demands generated from an increase in available image data. However, the development of these models with respect to earth observation tasks has lagged behind the progress in computer vision

ML. The specific challenges that arise from using ML for EO tasks have yet to be fully overcome (Atkinson & Tatnall, 1997; Song et al., 2019).

The most prevalent challenge when using ML for EO classification and detection tasks is a lack of training data available for specific platforms of EO imagery (Längkvist et al., 2016). While, there is an abundance of well-established, computer vision training datasets, these are often unable to be used for EO classification and detection tasks. This is because of characteristics such as the overhead and small-scale nature of EO data (Guo et al., 2018). Target objects in EO images are often small and decentralized compared to computer vision images (Azimi, et al., 2019). This issue, combined with the different overhead angle (as opposed to lateral) and added spectral complexity inherent in multispectral and hyperspectral images, makes the patterns learned by an ML for EO algorithm fundamentally different from those trained on computer vision images (Tripathy, 2019; Schweitzer & Agrawal, 2018). Additionally, common processing issues associated with EO data such as atmospheric interference and differences in sensors and lighting conditions make it difficult to develop an ML model which is useful across multiple datasets and locations (Etten, 2018).

1.5. Research Objectives

The purpose of this project is to develop a single CNN-automated algorithm to detect and classify different roof types in rural Africa based on their spectral and geometric characteristics using WorldView-2 and QuickBird multispectral satellite imagery. This algorithm aims to be flexible enough to be used for different countries and time-periods in Africa, as well as for imagery from different satellite sensors. If this can be achieved it may be possible to use EO data to monitor certain SDGs, which would help to reduce data gaps associated with decadal census enumeration.

In addition to the potential humanitarian benefits of using ML for roof type detection, it also has promising implications for the further development of ML techniques being applied to the processing/analysis of the exponentially increasing store of earth observation (EO) data.

Research Questions:

- Can machine learning compete with software-based detection and classification of roofs to improve accuracy of roof detection and classification, while maintaining the flexibility required to be used in different areas of rural Africa?
- How specific (in resolution, location, and size) do training data have to be to achieve an accuracy of above 80% on the target dataset?
-
- Can machine learning be used to detect changes in the number and type of roofs in rural African settings?

1.6. Study Area

The study area for this project is Mbola, a mostly rural area residing in the Tabora region of Tanzania. Mbola contains only a few small villages, six of which were part of the Millennium Villages Project (MVP) and received resources and education for the improvement of agriculture, health, and other community development initiatives (UNDP, 2007). The socioeconomic improvements stemming from

these efforts, especially those seen in asset-ownership, make Mbola a good candidate for the exploration of changes in roof material extracted from satellite imagery (Mitchell et al., 2018; Jean et al., 2016).

The primary datasets for this project consist of high-resolution, multispectral and panchromatic reflectance data of Mbola, Tanzania from 2007 and 2012 (both acquired from Digital Globe). The QuickBird 2007 dataset includes a panchromatic layer of 0.6 meter resolution and a multispectral layer of 2.4 meter resolution. The WorldView-2 2012 dataset includes a panchromatic layer of 0.5 meter resolution and a multispectral layer of 2.2 meter resolution. Mbola resides in the Tabora Region of Tanzania, which is a mostly rural area with a few small villages consisting of houses with primarily thatched roofs or metal sheet roofs.

A supplementary dataset from the Bulawayo district in Zimbabwe was used to test the model's flexibility. This 2019, Worldview-2 multispectral dataset's imagery was pre-pansharpened to a 0.31 meter resolution and included four spectral bands. The Zimbabwe dataset also consists of metal and thatch roofs, with the additional roof types of asbestos, tile, and roofs under construction. This dataset came with its own set of ground truth point shapefiles for each type of roof.

The full extent of both Tanzania and Zimbabwe datasets can be seen in Figure 1.

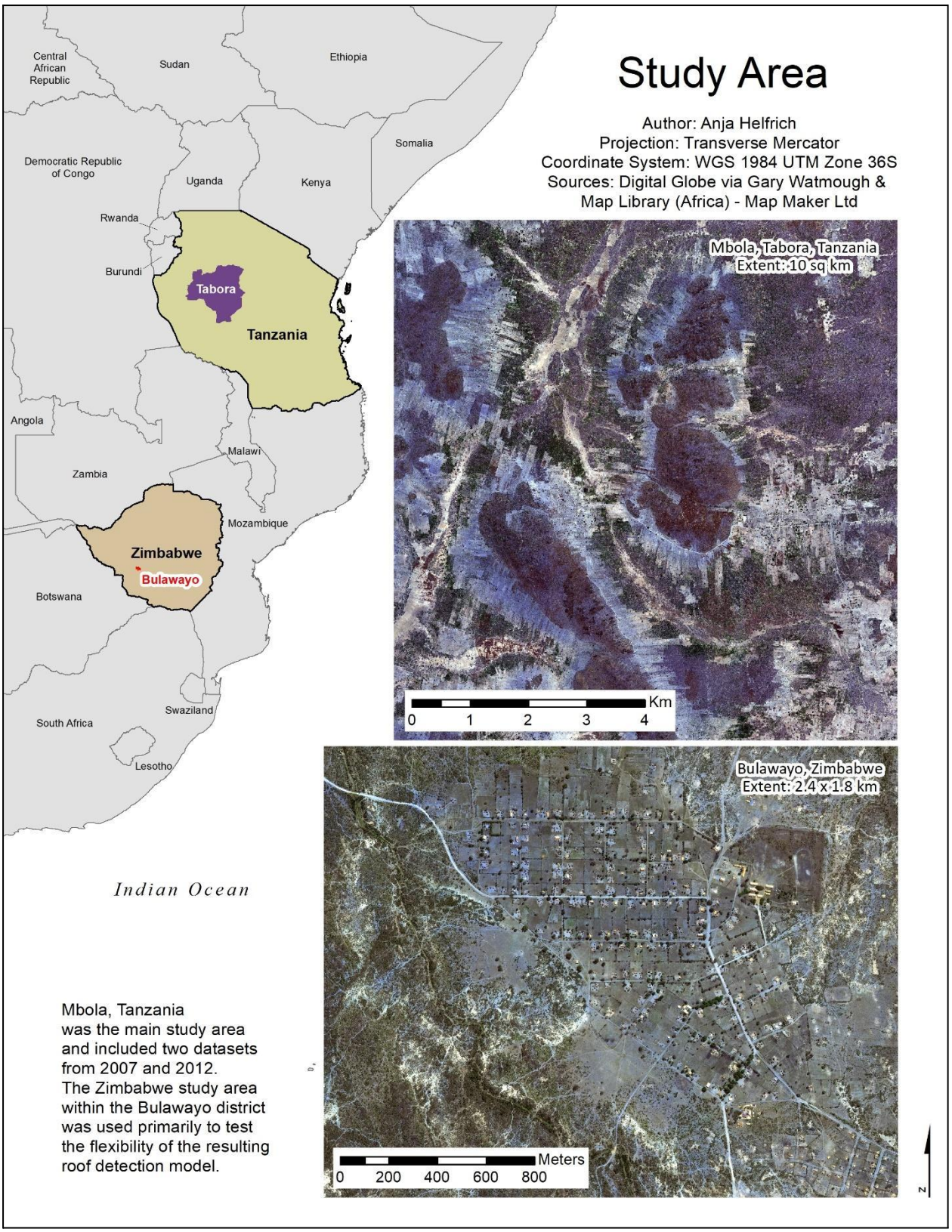


Figure 1: Map of study area, showing the full, true color extent of datasets used (Source of basemap: Map Library, 2007).

Four villages from the Mbola dataset were chosen for testing and creating ground truth data for validation (Figure 2). The entire Zimbabwe dataset was processed and tested since there is ground truth data for its entire extent.

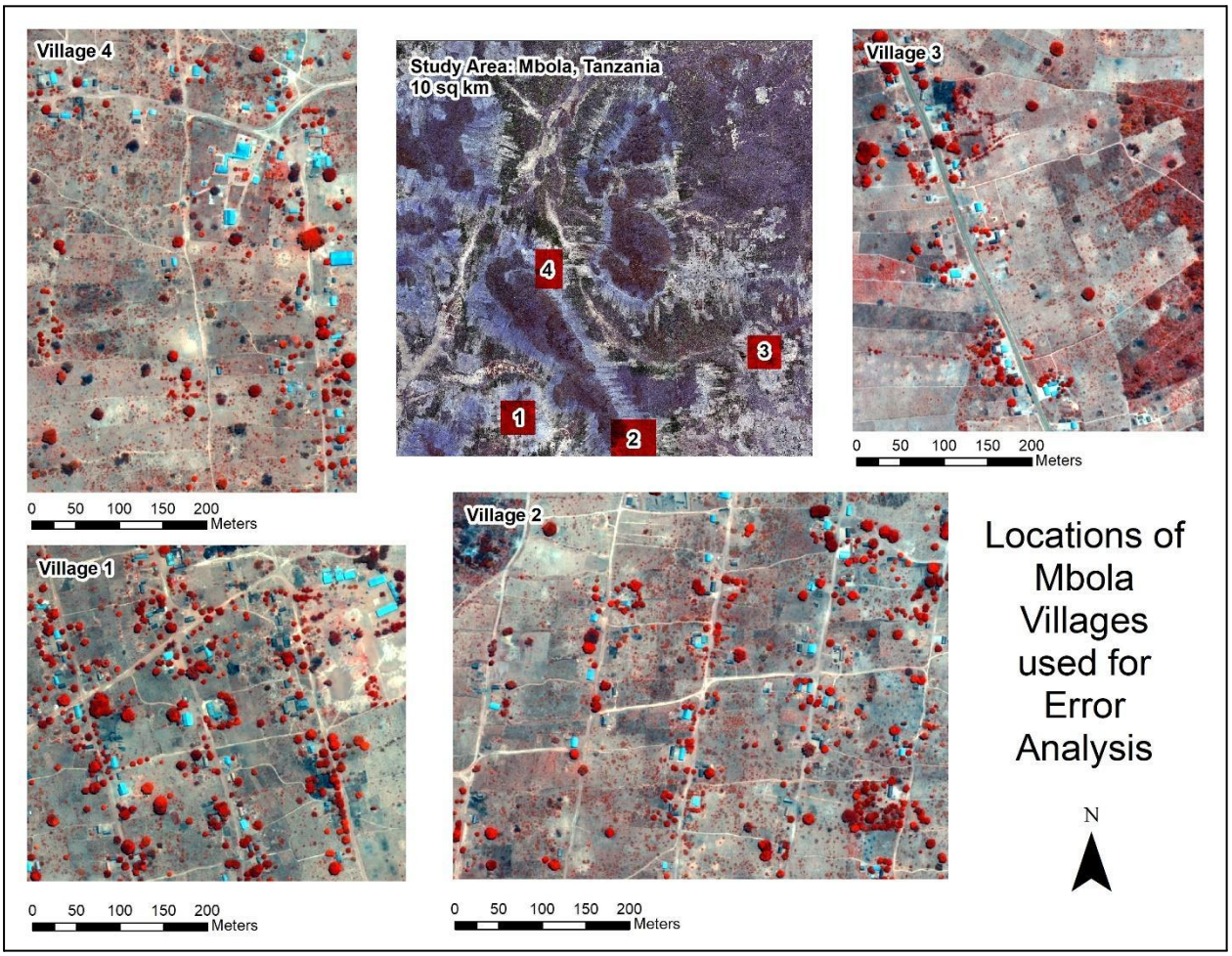


Figure 2: Location of Mbola Villages used for Testing and Validation. Ground truth data was manually created for these villages.

2. Methodology

2.1. Technical Requirements

The main technical requirements for this project included ArcGIS Desktop Software and a Python coding environment in Google Collaboratory. ArcMap and Arcpy were used primarily for pre and post-processing of data (ESRI). All Python coding was carried out in a Google Collaboratory Notebook environment for the freely-available GPU processing and the ability to work quickly on a local machine. Keras and TensorFlow were chosen as the main open-source coding packages necessary for ML capabilities because of their high-level API functionality, popularity in the ML community, and subsequent extensive documentation (Abadi et al., 2015; Chollet, 2015; Nguyen et al., 2019).

2.2. Pre-Processing

The Mbola dataset was split into nine subsets for ease of processing. The main step in pre-processing involved combining the spatial resolution of the panchromatic data with the spectral resolution of the multispectral data (Figure 3). This process is called pansharpening and was done using a python script created by Michalitsianos, 2019. There are several different techniques for pansharpening, all of which were tested and visually assessed to choose the most successful result, the Brovey technique (see Technical Report Section 4). A false color composite of NIR = R, Red = G, and Green = B was used throughout the project for clear visualization of roofs (Figure 3, Pansharpened).

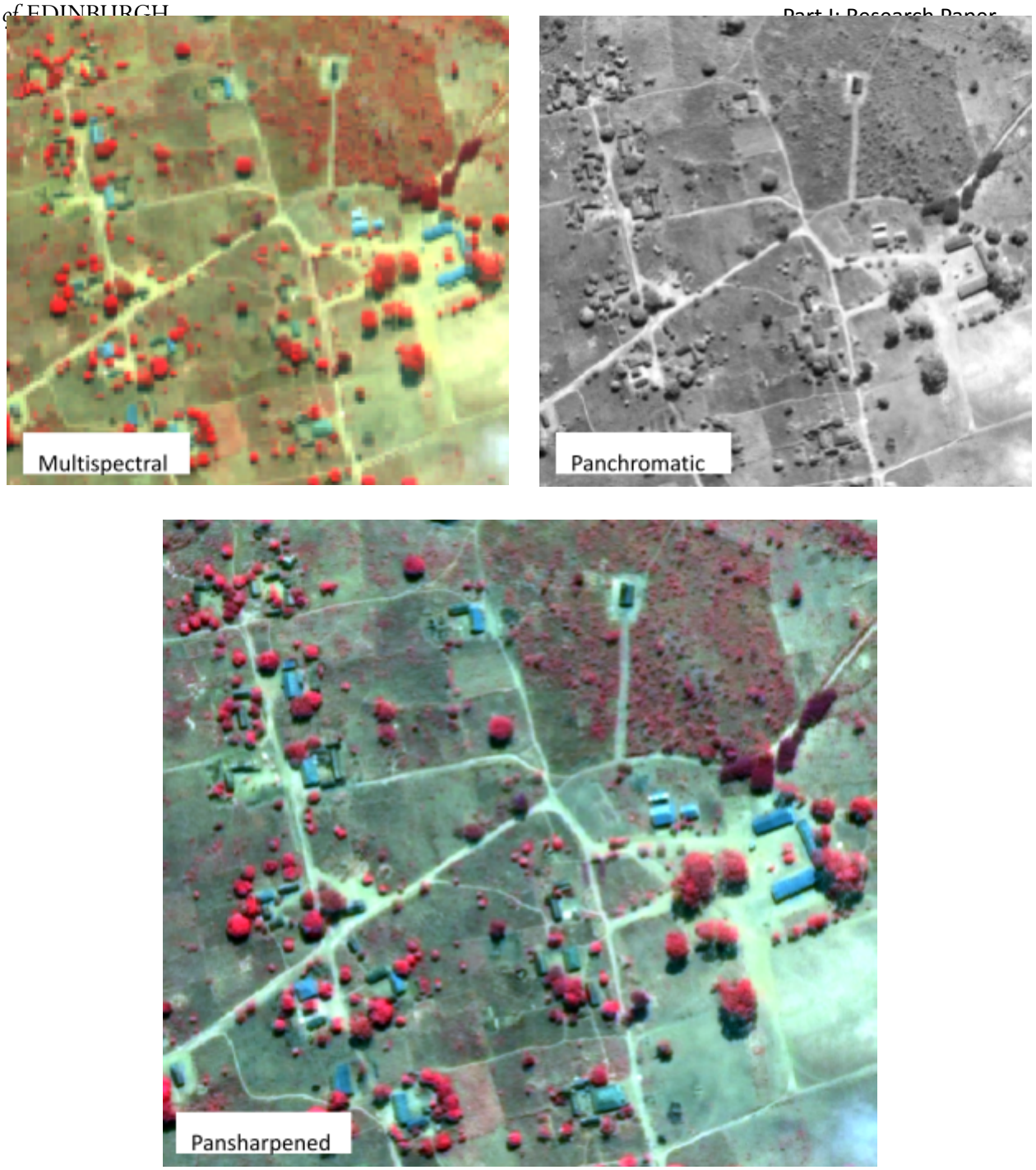


Figure 3: Example of Brovey pansharpening process on a subset of data from the 2007 Mbola dataset, with the two top images as inputs, and the bottom image as the output. Multispectral and pansharpened images are shown in as a false colour composite (NIR = R; Red = G; Green = B) for ease of visual roof detection.

2.3. Creating Training Data

ML tasks often require large amounts of training data (Schweitzer & Agrawal, 2018). While there have been ML training datasets created for the specific task of detecting roofs in satellite imagery, these datasets are specific to areas of the world where the roof and background landscape characteristics differ widely from those in Africa (Figure 4). Therefore, these datasets could not be successfully used to train the model.

Figure 4: Example of EO training datasets for building detection. Both datasets are Worldview-3 and similar in spatial resolution to that of the Mbola dataset. ***a)*** *Pre-disaster image, part of xView2 disaster classification challenge (Ritwick et al., 2019)* ***b)*** *Building footprint data from Hungary, part of a weakly-labeled dataset of 669 footprints (Benedek et al., 2010). Notice background characteristics such as concrete roads, and different roof and tree types from those found in Mbola imagery (Figure 3).*

Custom training data was therefore created using ArcMap's 'Training Sample Manager' and 'Export Training Data For Deep Learning' tools (ESRI).

The southwest corner subset of the 2012 Mbola dataset was chosen for the extraction of training samples because it was cloud-free, had a variety of background characteristics, and had an abundance of thatch roofs (of which there is a lack of in other parts of the dataset).

A set of rules were established for choosing/excluding roofs from the training dataset (see Technical Report Section 5). After digitization, samples were labeled as Metal or Thatch depending on their spectral characteristics (Figure 5).

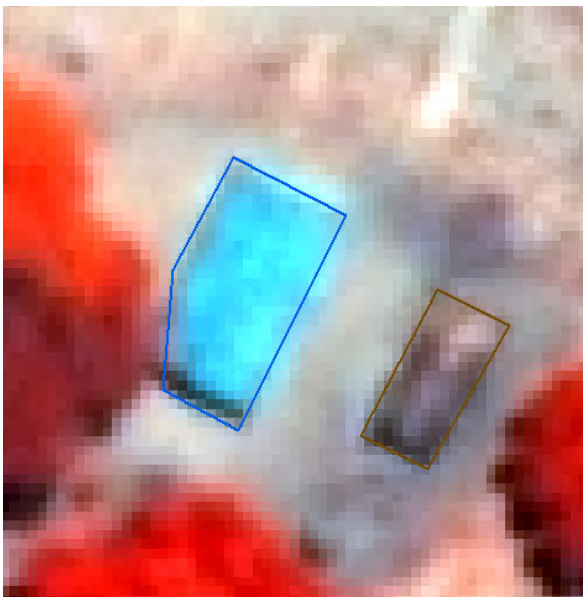


Figure 5: Example false color composite with metal (blue) and thatch (brown) training samples selected using Training Sample Manager in ArcMap. Note the inclusion of shadows and exclusion of trees and pixel bleeding.

Reflectance scatter plots of metal vs thatch roofs were produced to ensure sample quality and relative consistency in choosing samples (Figure 6). High-quality data aims to have small 'within-class' scatter and large 'between-class' separation (Cheng et. Al., 2018). These scatter plot comparisons of reflectance values were analyzed to determine the spectral differences between thatch and metal roofs, and to further assess whether outliers were 'bad' samples or unique examples of roofs that would be beneficial for the ML model to 'see' a larger diversity of true roofs, which would increase the model's flexibility across datasets (Li et al., 2019). The scatter plots are also a good way to visualize which spectral bands are the most significant in distinguishing between thatch and metal roofs.

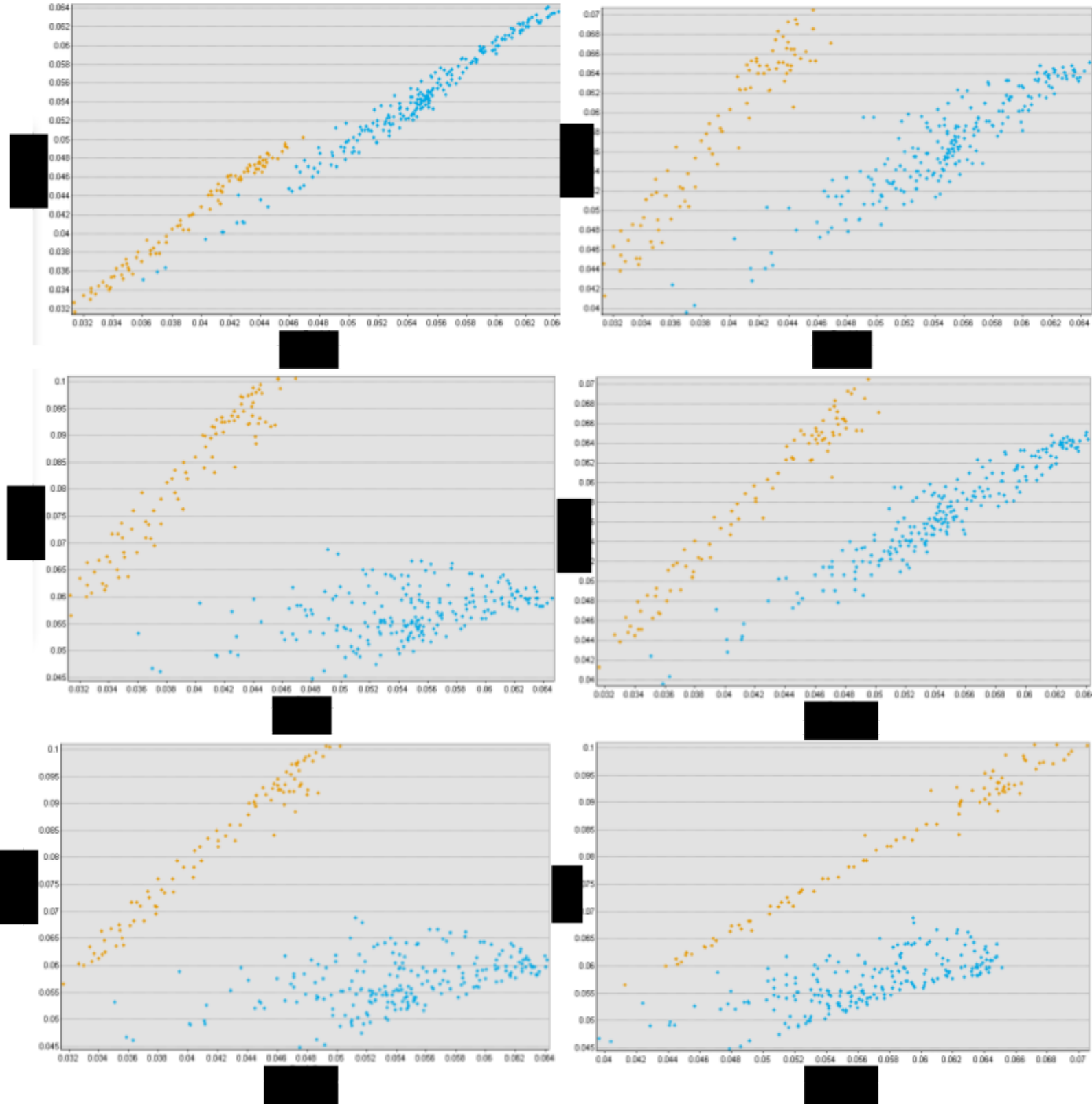


Figure 6: Spectral scatter plots of metal (blue) and thatch (brown) samples. Note the mostly separate clusters representing unique spectral signatures.

After quality assessment, training samples were 'merged' by type and exported as shapefiles for input into the 'Export Training Data for Deep Learning Tool' (see Technical Report 4).

The output of this tool includes directories with the clipped 30x30 pixel sample images and corresponding xml files with label and bounding box information in the Pascal VOC style of annotation (Everingham et al., 2015). The size of training sample images was chosen based on the average size of buildings and distance between buildings so that anywhere from 0-2 buildings were likely to fall within the same training sample. The ideal size would be to include 0-1 buildings, but a 30x30 pixel sample size already pushes the limits of what is acceptable to train a CNN, and any smaller would have not provided enough contextual information for the CNN classification (Zhang et al., 2017).

Since there are significantly less thatch roofs compared to metal in both Mbola datasets, and CNN models benefit from training on equal amounts of classes, additional subsets of the 2012 Mbola dataset were used to obtain an adequate amount of thatch samples (Li et al., 2019).

2.4. Augmentation

To increase the amount as well as the diversity of training data for these purposes, different types of augmentation were performed on array-converted training samples using Python (Appendix I) These included a horizontal flip, a ninety-degree clockwise rotation, and later a histogram stretch based on the range of spectral reflectance values in the Zimbabwe dataset to make the model more robust for use on datasets from different African countries and sensors (Campbell, 2019; see also Figure 7).

Augmentation has been shown to be especially useful for increasing the size of training data for EO ML tasks because of the overhead nature of these images (Shorten et al., 2019; Miyamoto et al., 2018). The benefit of EO data in a ML context is that the overhead view angle makes it acceptable to rotationally and inversely augment data that would produce directionally inaccurate (i.e. ‘upside down’) image samples for computer vision tasks. This is especially helpful since there is a lack of pre-labeled/processed EO training data compared to computer vision tasks.

Randomly scaling the values of these images has also been shown to improve the robustness of a model when testing images come from different sensors with different atmospheric and lighting conditions (Etten, 2018; Zhu et al., 2017).

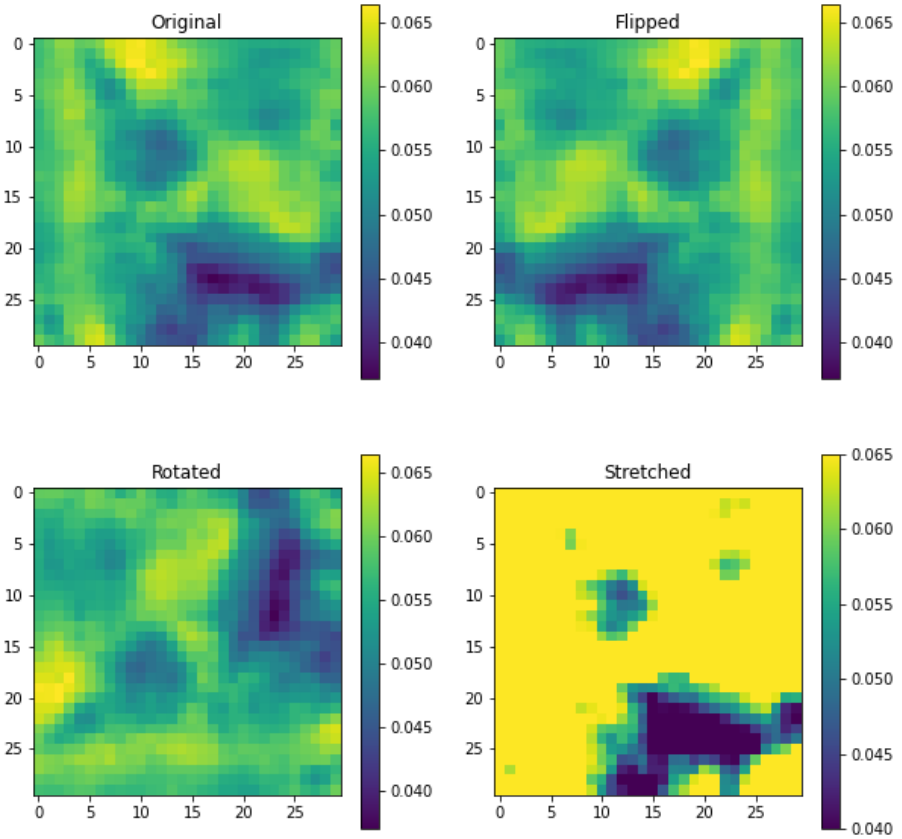


Figure 7: Example of augmented 30x30 pixel training sample with the stretched reflectance value axis modified to match that of the other samples to visually show the difference (Appendix I). The actual stretched reflectance values range from 0.04 to 1.31 which is equal to the range of values in the Zimbabwe dataset.

Since thatch roofs were scarce, augmentation was especially useful for this particular dataset (Zhu et al., 2017). No augmentation was performed on the background training samples to preserve the range and diversity of background features and because of their relative abundance compared to thatch and metal training samples.

2.5. Building and Training the Convolutional Neural Network

The decision to build a simple classification model from scratch instead of using a pre-constructed model architecture stems from the small training data size and subsequent need to avoid over-fitting and from the difficulty in converting such a model to one that can take in four spectral bands. This decision was supported by the fact that the features to be learned are relatively simple, low-level patterns (squares) of relatively uniform scale and angle (Salcedo-Sanz, et al., 2020). The final CNN Classifier architecture consists of three convolutional layers, a pooling layer, two dropout layers and two dense layers (see Technical Report Section 6).

The creation of an Object Detection model was also attempted. However, CNN object detection is a much more complex task than classification alone, and project time constraints therefore made this an unrealistic option (see Technical Report Section 7; Appendix III).

The final model was trained on 3,737 images or 90% of 4,152 samples (see Appendix II). The remaining 10% were set aside for preliminary testing of model accuracy. Roughly equal amounts of each class (No Roof, Metal, and Thatch) were represented. The final model trained on this data for 600 epochs, with data being 'shuffled' between each epoch. The model was set to stop training after the loss ceased to decrease using the Keras Early Stopping tool with a 'patience' value set to 30 epochs (TensorFlow Core v2.3.0). This was done to prevent over-training of the model, where the model 'memorizes' the training dataset. Overtraining can severely limit a CNN's ability to be used on any data it has not trained on (Zhang et al., 2017). An example of the training process (run on less epochs for better visualization) can be seen in Figure 8.

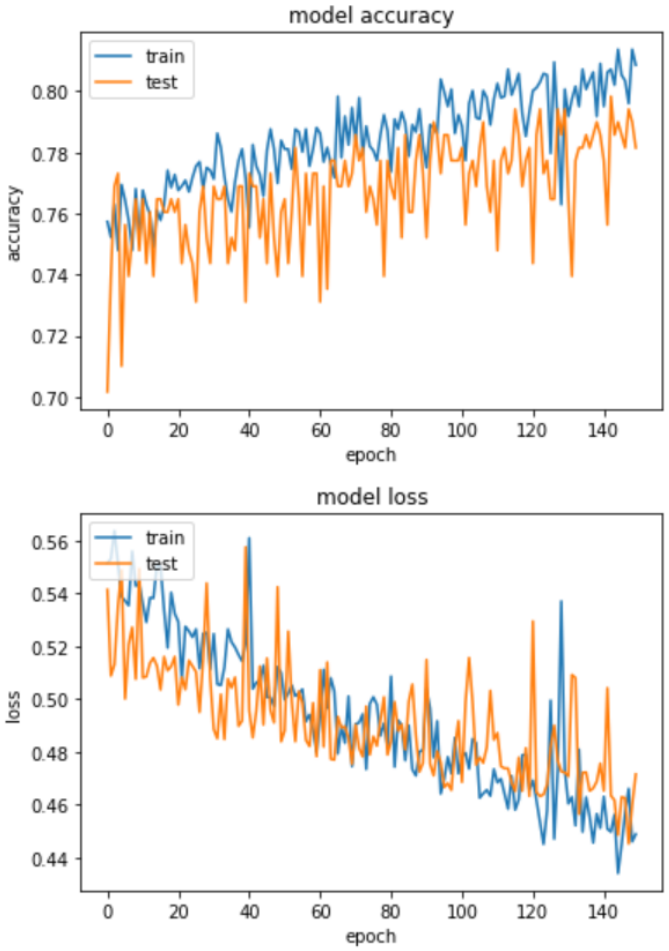


Figure 8: An example of the training process and the progress of loss and accuracy over a certain number of epochs. Notice the increase in accuracy and the decrease in loss. As soon as the loss starts to level out or increase, training should be stopped to avoid overfitting.

The final trained model had an accuracy of 90% and a loss value of 0.1. All datasets were run through this model (see Appendix IV). The outputs were converted from array to tiff format and included probability masks and a classification masks for each dataset (the four Mbola images for each year and the entire Zimbabwe dataset). Probability masks consist of three different bands representing the different classification probabilities (from 0-1) (see Technical Report Section 8). Label masks consist of one band representing the classification based on the maximum probability value in that particular 30x30 window. These resulting masks were then exported for post-processing and validation in ArcMap. The entire process of creating, training, and implementing this CNN model can be seen in Figure 9.

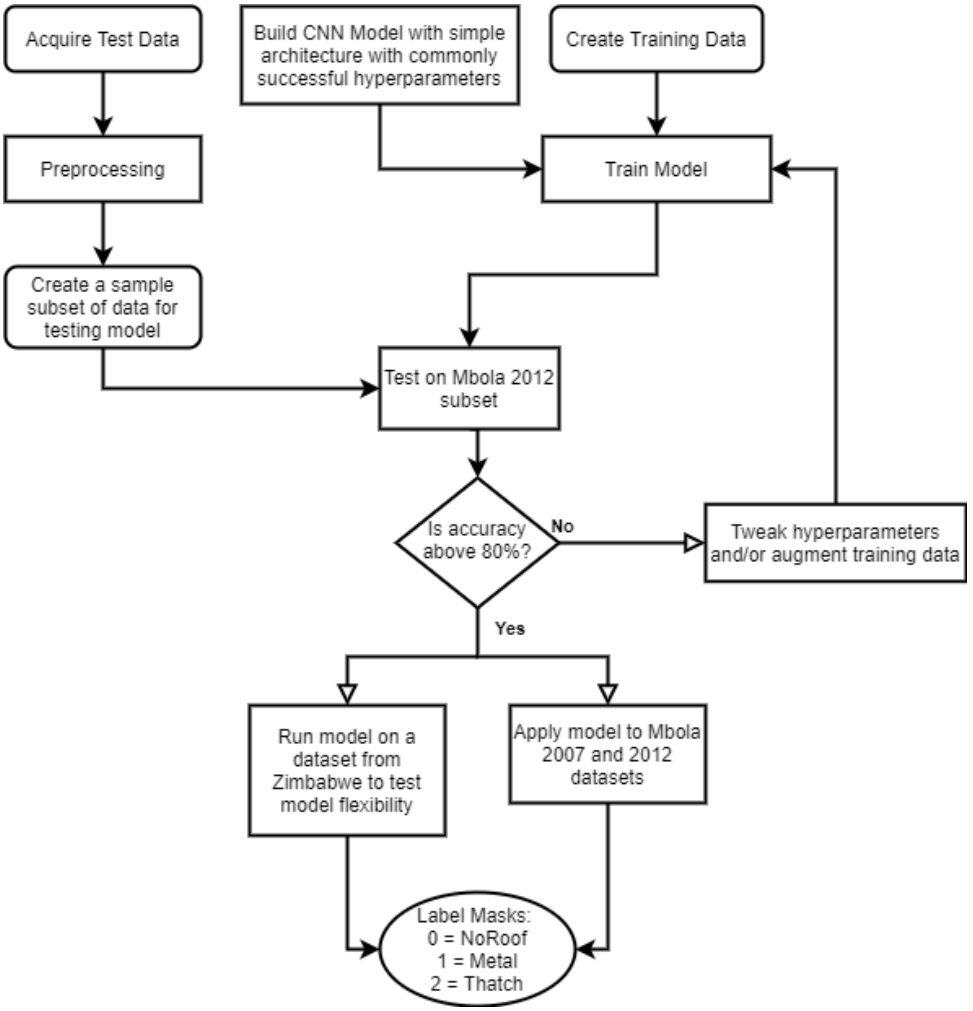


Figure 9: Total workflow for pre-processing, creating training data, training CNN Classifier and producing results for later combination with segmentation outputs and validation testing and analysis.

2.6. Segmentation

It was deemed necessary to narrow down the precise location of classified roofs beyond the 30x30 pixel masks produced by the CNN Classifier. In lieu of an object detector, the next best solution was

to combine the relatively accurate CNN Classifier output masks with a segmentation carried out in ArcMap (ESRI).

Two different segmentation workflows were tested in ArcMap with the more successful process chosen (see Technical Report Section 9; Appendix V). The chosen workflow included Training an ISO Classifier for classification using three bands of the original imagery (Red, Blue, and NIR). The output of this classification was then reclassified based on a visual analysis of which classes best represented metal and thatch roofs.

2.7. Post-Processing and Validation

Post-processing of the results involved combining the CNN Classifier and Segmentation outputs (see Technical Report Section 10; Appendix VI). Since both outputs are of the same size and classified in the same way (0 = No Roof, 1=Metal or Thatch), the Raster Calculator tool in ArcMap could be used to multiply these outputs. If either input were zero, the result would be classified as No Roof. If both inputs were equal to one, the result would be classified as Roof. The outcome was a series of predicted metal roof and thatch roof maps.

These prediction maps were then converted to polygons. Polygons classified as No Roof were filtered out so that only building polygons remained. Buildings less than two square meters were discarded and buildings within four meters of each other were aggregated. This was done to smooth out noise from the segmentation process and more accurately represent buildings.

The Intersect tool was used to perform 'Polygon in Polygon' tests to identify buildings that were located and classified correctly as well as buildings that were located but misclassified as the wrong roof type. Based on the total number of predicted and true buildings within a certain class, false positives and false negatives could then be calculated. These values were inserted into confusion matrices, from which various accuracy metrics could be calculated.

The same validation process was used for the Zimbabwe dataset. However, since the ground truth data was in point format, circular buffers (with a radius of 4 meters) were created from these points based on the average size of buildings. These circular buffers were used as ground truth polygons.

2.8. Accuracy Assessment

Three accuracy measures were calculated based on the resulting confusion matrices and common Object-Based Image Analysis (OBIA) accuracy assessment techniques. User's and Producer's accuracies were calculated using the following equations:

1) $$UA = User's Accuracy = Recall = \frac{TP}{TP+FN}$$

2) $$PA = Producer's Accuracy = Precision = \frac{TP}{TP+FP}$$

where TP represents True Positives, FN represents False Negatives, and FP represents False Positives. These measures help to ascertain if the model tended more towards over-prediction or under-prediction and were also used to calculate the F-measure:

3) $$Fmeasure = \frac{2*Precision*Recall}{Precision+Recall}$$

The F-measure, or F-Score, is a commonly used OBIA accuracy metric for assessing overall accuracy and is defined as the harmonic mean of precision and recall (Vafeiadis et al., 2015). This metric is used in a summary fashion since there is no way to calculate True Negatives for object-based detections and therefore no way to calculate the traditional overall accuracy used in segmentation classification assessments (Cai et al., 2018; Radoux & Bogaert, 2017).

Agreement maps were created to assess the relative success of the segmentation process vs the CNN model process. Raster Calculator was used to add the outputs from these two methods together to visualize where the outputs agreed and disagreed.

3. Results

3.1 Mbola, Tanzania: Accuracy Assessment

The average overall F-score for detecting and classifying roofs in Mbola was 68% for metal roofs and 9% for thatch roofs. The algorithm consistently performed better on the 2012 dataset, with a higher average F-score of 73% for metal roofs and 14% for thatch roofs. The highest F-score obtained was 85% for metal roofs in the 2012 Village 4 dataset (Figure 10). Metal roofs also produced more homogenous User, Producer and F-measure accuracies across different villages and years (Table 1). The classification and detection of thatch roofs was significantly worse with generally low F-scores ranging from 0% to 16% and erratic User and Producer Accuracies ranging from 0-100% (Figure 11).

Producer's accuracy was generally higher except in the case of the 2012 thatch dataset where the low number of false negatives contributed to a higher User's Accuracy (see Appendix VII).

Table 1: User's and Producer's Accuracies as well as F-Measure for all Mbola metal and thatch datasets.

	Producer's Accuracy		*User's Accuracy*		***F-Measure***	
2007	Metal (%)	Thatch (%)	Metal (%)	Thatch (%)	Metal (%)	Thatch (%)
Village 1	81.3	0.0	47.4	0.0	**59.9**	**0.0**
Village 2	98.9	25.0	47.0	3.7	**63.7**	**6.4**
Village 3	76.0	0.0	47.0	0.0	**58.1**	**0.0**
Village 4	64.0	9.5	79.0	11.8	**70.7**	**10.5**
2012						
Village 1	90.2	9.0	55.7	36.0	**68.9**	**14.4**
Village 2	93.6	7.6	66.5	26.3	**77.8**	**11.8**
Village 3	78.4	8.8	48.7	100.0	**60.1**	**16.2**
Village 4	100.0	8.0	73.5	47.4	**84.7**	**13.7**

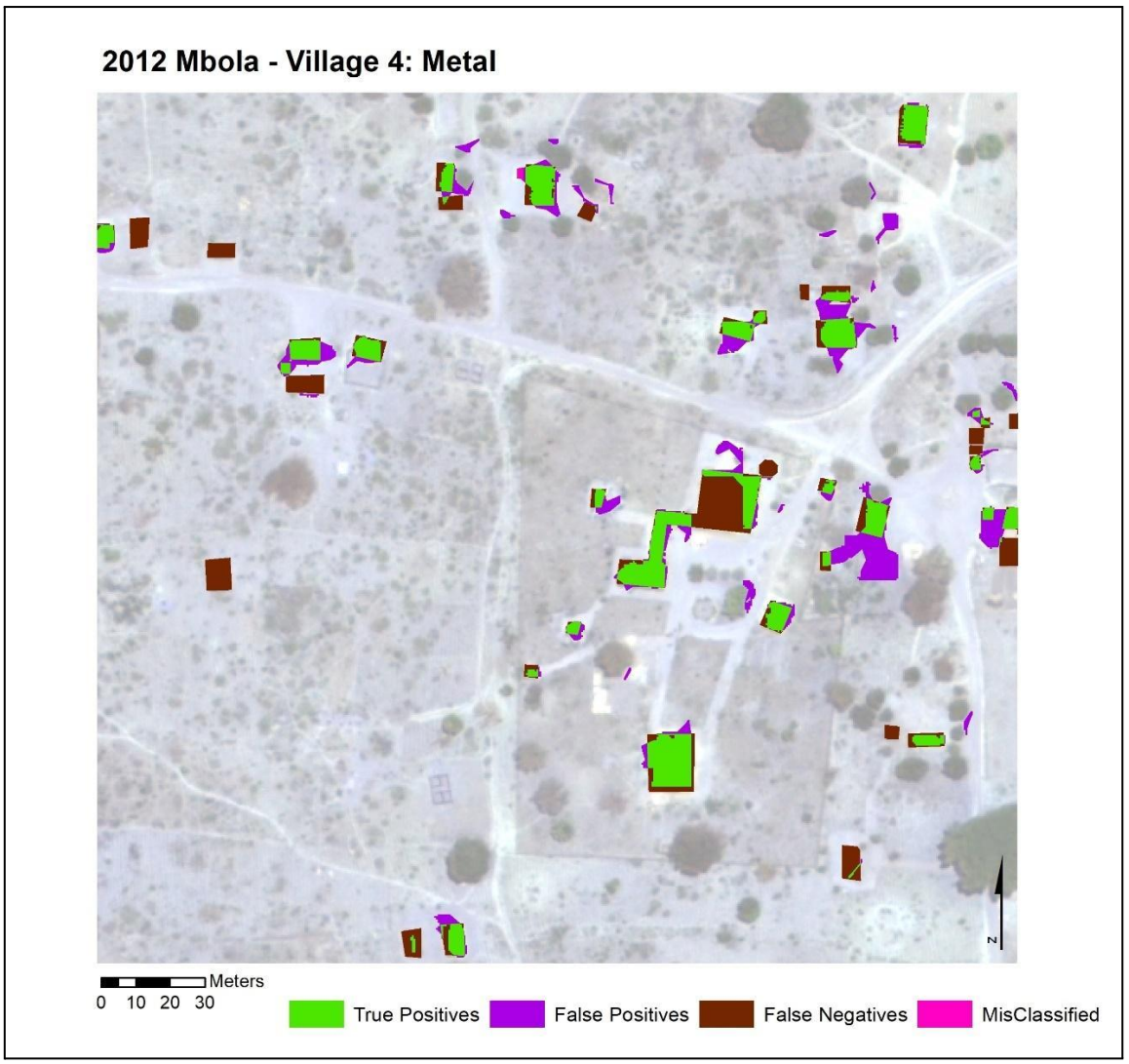


Figure 10: An example of the 2012 Village 4 metal predictions where the model performed well with an overall F-score of 85%. Notice that false positives are often adjacent to true positives. Additionally, note the low number of misclassified buildings, which is representative of all tested datasets (see Appendix VIII for all map results).

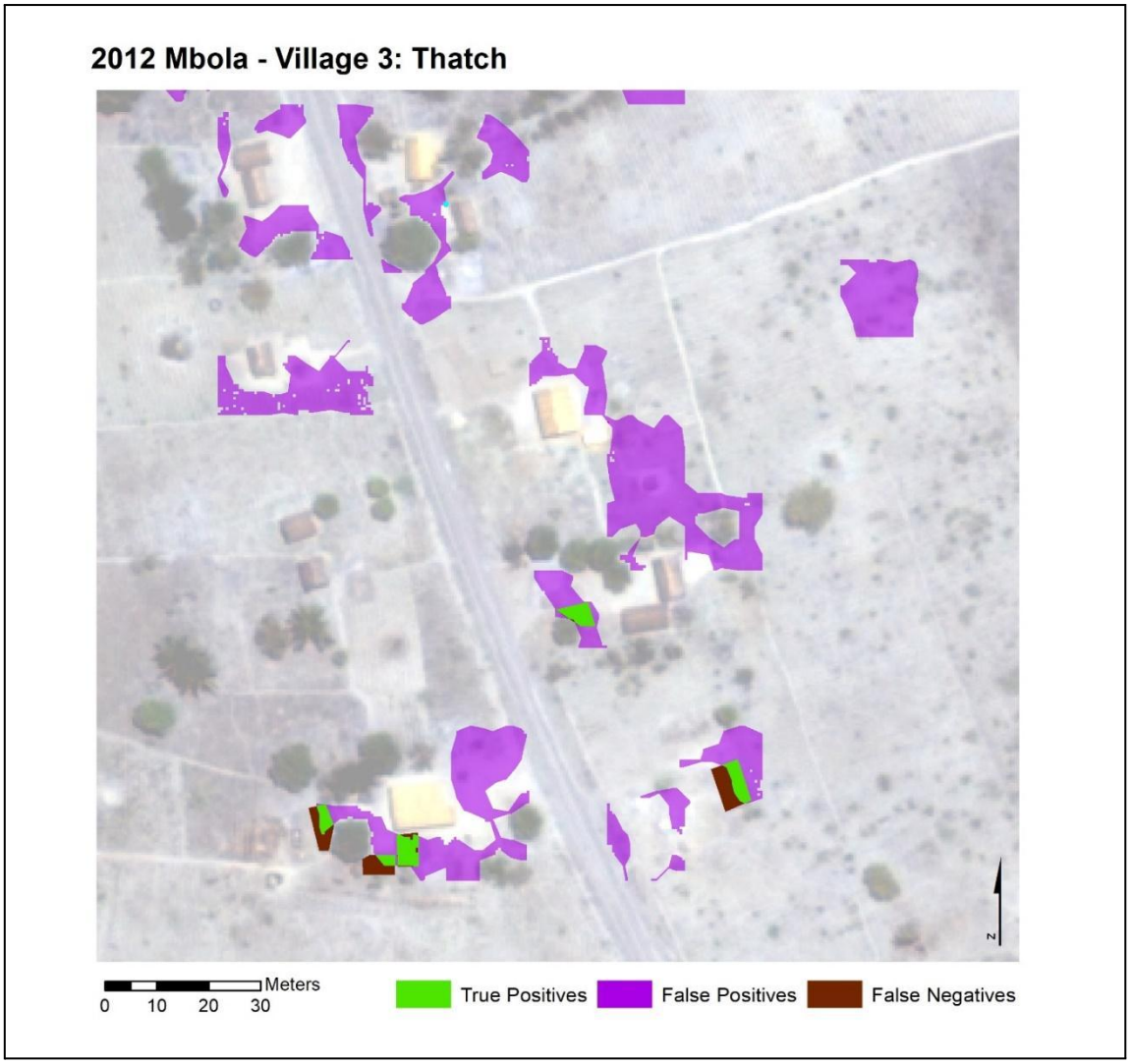


Figure 11: An example of 2012 Village 3 thatch predictions where the model performed poorly. However, this dataset had the highest accuracy across thatch predictions with an overall F-score of 16%. Note the large areas of false positives which were the main contributor to poor accuracy.

3.2. Mbola, Tanzania: Change Analysis

The predicted change in number and type of roofs between 2007 and 2012 indicates an increase in thatch roofs and a larger increase in metal roofs. This pattern generally reflects the actual changes seen in Mbola between 2007 and 2012 (Figure 12).

Figure 12: Charts of Predicted vs Actual Changes in Mbola Roofs between 2007 and 2012.

Change predictions are generally consistent in underpredicting the increase in metal roofs (except for Village 2) and overpredicting the increase in thatch roofs, especially in the case of Village 2 where the actual change shows a decrease in thatch roofs.

3.3. Analysis of Agreement Between Segmentation and CNN Model Outputs

The comparison between the CNN Masks and Segmentation shows where the two methods agree and have subsequently classified a roof and where they disagree or where one of the models is in error (Figure 13). This agreement map reveals the expected blocky nature of the CNN output masks and the noisy nature of the segmentation predictions. It can also be seen from this map that the thatch segmentation seems to have classified some roads as thatch roofs.

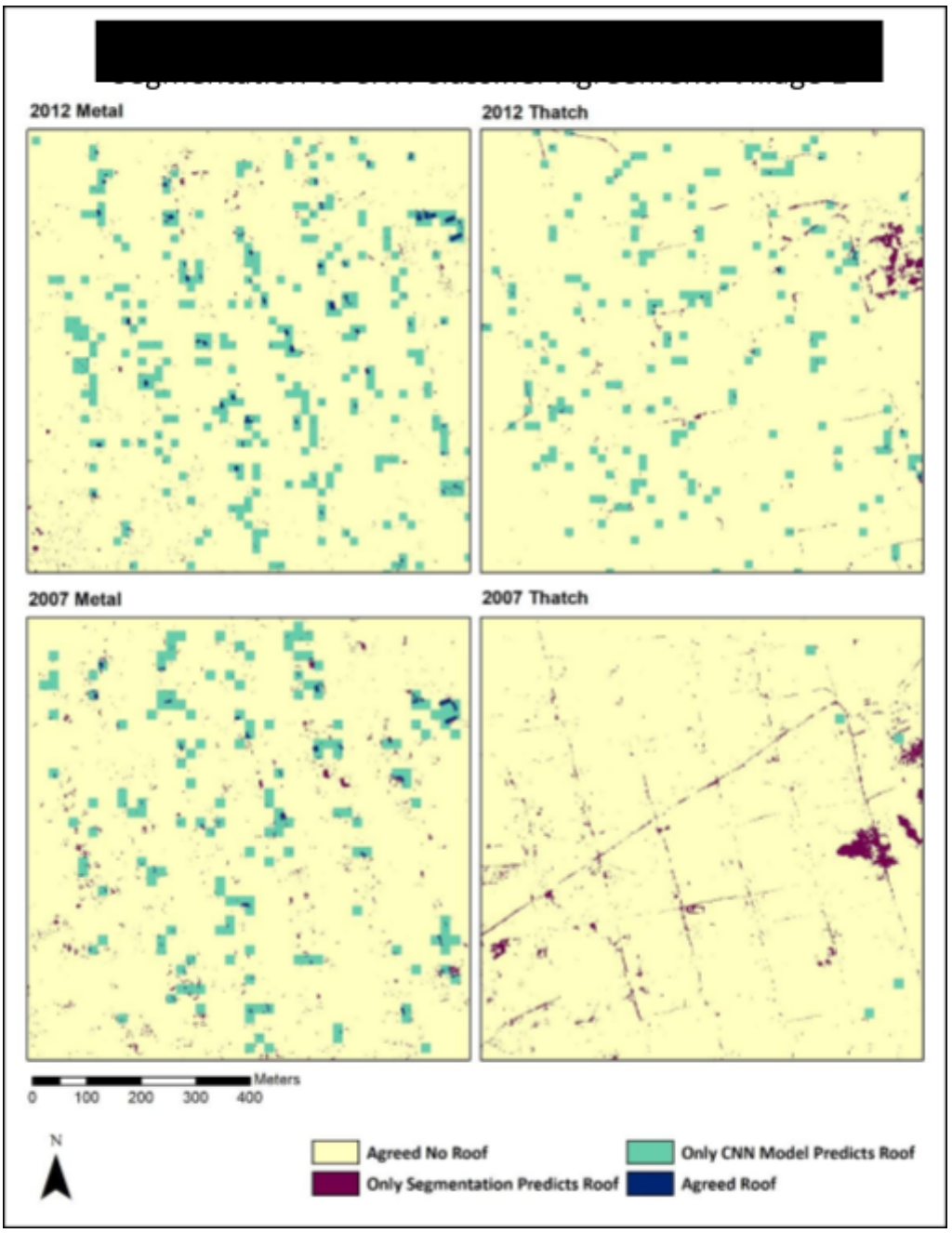


Figure 13: Example of agreement map between CNN Model and Segmentation Outputs for 2012 Mbola Village 1.

3.4. Zimbabwe Results

Model accuracy for metal drops significantly when applied to the Zimbabwe dataset with an F-score of 37.9% (Table 2). If compared average F-score in the Mbola metal dataset this is a drop of over 30% in accuracy. Interestingly, the model accuracy for thatch increases by more than 15% compared to the Mbola thatch dataset. Producer's accuracy is higher for metal roofs and comparable for thatch roofs.

Table 2: User's and Producer's Accuracies as well as F-Measure for the Zimbabwe metal and thatch datasets.

Zimbabwe	*Producer's Accuracy (%)*	*User's Accuracy (%)*	***F-Measure (%)***
Metal	84.4	24.4	**37.9**
Thatch	26.1	28.6	**27.3**

A visualization of how Zimbabwe metal and thatch roof detection compares can be seen in Figure 14. This comparison shows abundant false negatives due to the different roof types present in Zimbabwe that the model did not have training samples for.

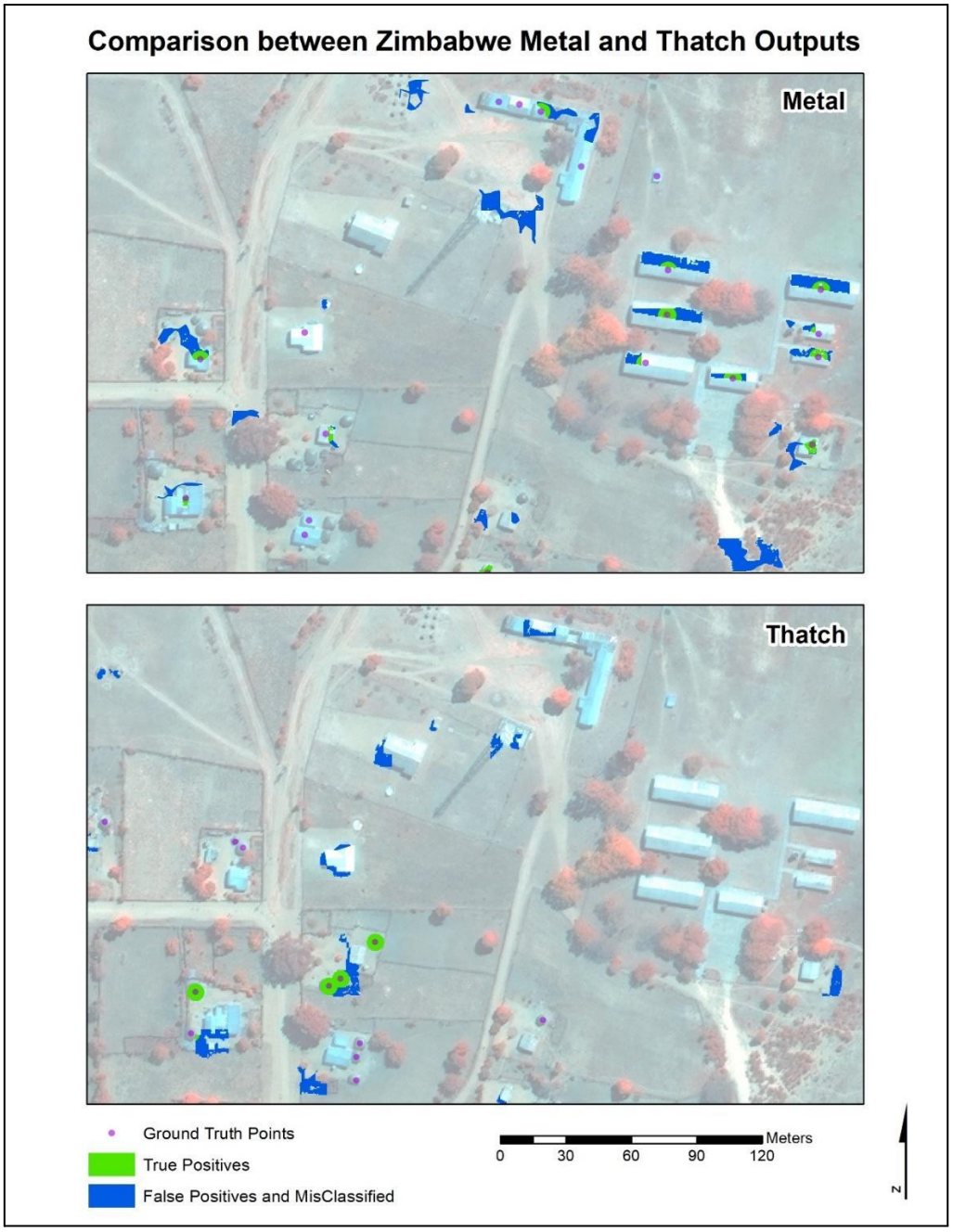


Figure 14: Comparison of metal and thatch model outputs for Zimbabwe.

The agreement maps between CNN and Segmentation outputs can be seen in Figure 15.

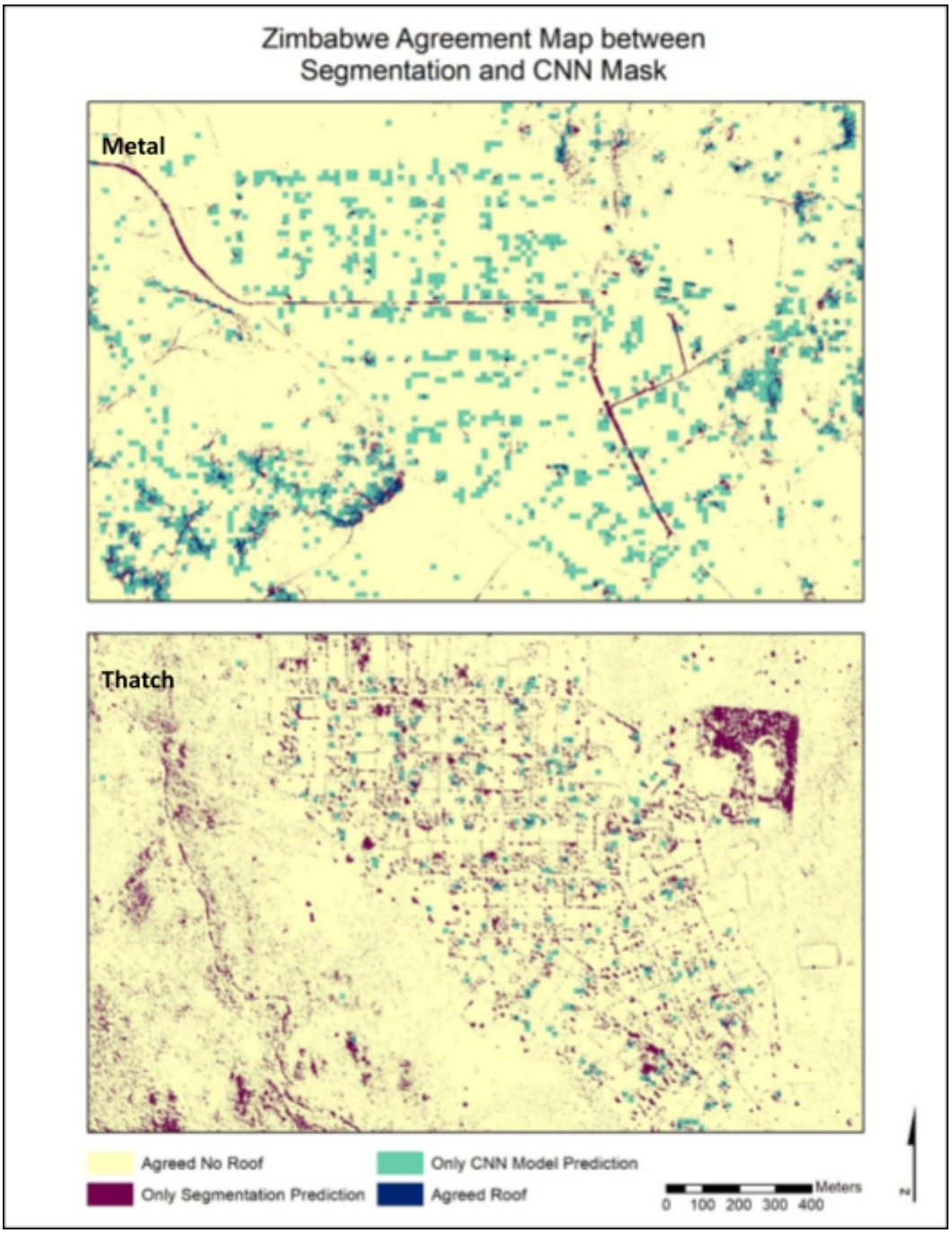


Figure 15: Comparison of metal and thatch agreement maps for Zimbabwe.

The agreement map for Zimbabwe shows an area of false positives in the southwest corner of the metal roof map. It also shows a much noisier segmentation for thatch roofs as opposed to the sparsely predicted thatch roofs shown by the CNN Model masks. For metal roof predictions, it shows the same issues of misclassifying roads as roofs, as did the thatch segmentation in Mbola.

4. Discussion

4.4. Research Questions Revisited

- While the highest accuracies achieved by the final model are comparable to those of software-based detection and classification of roofs, the average overall accuracy is significantly lower for both metal and thatch detections. The same issues of cross-country flexibility and less accurate thatch detection arise as occur with software-based methods (Campbell, 2019). For the computer vision field, ML has proven to be an accurate and efficient way of classifying, localizing, and detecting objects, but much work has yet to be done to make it a viable solution for similar tasks using satellite imagery. There is also difficulty in conducting proper error analysis since CNN's are essentially black box models which makes it difficult to pull out specific patterns that the model is picking up. This also makes it difficult to improve the model without a lot of trial and error experiments.

- If large and diverse amounts of training data are not available, training data must be extremely specific to the imagery to be classified in order to achieve an accuracy above 80%. This is supported by the only model output above 80%, which was from the same location and time period that the CNN Classifier trained on. However, final results were affected by poor segmentation, so further evaluation of the CNN Classifier on its own is needed to fully answer this question.
-
- The final model was able to accurately predict general trends in roof type changes between 2007 and 2012. Change analysis shows an overall increase in buildings, with a larger increase in metal roofs. This is not surprising given MVP involvement in Mbola during this time period.

4.5. Sources of Error

There were many steps in creating and implementing this algorithm and therefore many sources of error. As expected, the model performed worse on datasets with different characteristics from the data the model trained on. The difference sensors between the 2007 and 2012 Mbola datasets in addition to the 3.6% cloud cover in the 2007 dataset may have had small effect on model performance.

Another probable source of error is any 'badly' labeled training samples. The quality of training samples is especially important when the training dataset is small, and even more important when using augmentation since any bad samples will be multiplied by the number of augmentations (in this case,
tripled). Any errors are further propagated in the training process since the model is exposed to the samples multiple times depending on the number of training epochs.

Similarly, since ground truth data for the Mbola villages was created manually in ArcMap, human digitizing errors are possible.

A large source of error, especially for thatch roofs, was the segmentation process. The segmentation outputs were generally noisy and created an abundance of false positives, which were not always filtered out by the CNN model where it was weak.

Finally, using the Aggregate tool in ArcMap to 'smooth' out errors created from the segmentation may have led to instances of combining two buildings into one, if particularly close together, or filtering out accurate predictions with the size threshold.

4.6. Future Developments

There are many ways in which the developed algorithm for Mbola roof detection could be improved. More expansive and diverse training data is needed across different resolutions, locations, and sensor platforms to increase ability of CNN models to be easily developed and implemented across different countries.

The CNN model hyperparameters could be fine-tuned more properly. At the moment, the standard techniques for developing ML models are to either use transfer learning or go through a long trial and error process of fine tuning a model for a specific purpose. More tools are therefore needed for the automation of fine-tuning hyperparameters, as well as for analyzing and visualizing patterns that CNN's are extracting to combat 'black-box model' issues.

The Object Detection model could also be greatly improved in order to finish its training and be able to use bounding box info for object detection as opposed to using segmentation to refine the locations of a 'moving window classifier'. Probability thresholds could also be used to further fine-tune classification outcomes.

5. Conclusion

While the developed algorithm to detect and classify roofs in Mbola, Tanzania did not meet the original accuracy and flexibility goals, the use of CNNs for extracting objects from satellite data still shows promise and has been successful in other locations. This study highlights the need for an increase in size and diversity of ML training data specific to African countries and Earth Observation (EO) tasks. Further development of ML analysis techniques is also needed to better understand the patterns extracted from data and to improve the efficiency of the training process. These developments are needed before CNNs can become a viable tool for the task of SGD monitoring in African countries.

6. References

Abadi, A., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jozefowicz, G., Jia, Y., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Schuster, M., Monga, R., Moore, S., Murray, D., Olah, D., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X. (2015) TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from: tensorflow.org

Atkinson, P.M. & A. R. L. Tatnall (1997) Introduction to Neural networks in remote sensing. *International Journal of Remote Sensing*. 18(4), pp. 699-709, DOI: 10.1080/014311697218700

Azimi, S.M., Vig, E., Bahmanyar, R., Korner, M., Reinartz, P. (2019) Towards Multi-class Object Detection in Unconstrained Remote Sensing Imagery. *Information Security and Privacy* pp. 150–165. Available from: https://dx.doi.org/10.1007/978-3-030-20893-6_10

Beegle, K., De Weerdt, J., Friedman, J., and Gibson, J (2012) Methods of Household Consumption Measurement through Surveys: Experimental Results from Tanzania. *Journal of Development Economics.* 98(1) pp. 3–18

Benedek, Cs., Descombes, X., and Zerubia, J. (2010) Building Detection in a Single Remotely Sensed Image with a Point Process of Rectangles *International Conference on Pattern Recognition (ICPR).* pp. 1417-1420, Istanbul, Turkey, August 23-26, 2010

Brownlee, Jason. (2019). A Gentle Introduction to Object Recognition With Deep Learning. *Deep Learning For Computer Vision.* Available at: https://machinelearningmastery.com/object-recognition-with-deep-learning/

Cai, L., Shi, W., Miao, Z., Hao, M. (2018) Accuracy Assessment Measures for Object Extraction from Remote Sensing Images. *Remote Sensing*. 10(2) pp. 303. Available from: https://dx.doi.org/10.3390/rs10020303

Campbell, Amy (2019). *Developing an automated algorithm to detect rural house roofs in Kenya using fine spatial resolution images*. School of Geosciences. University of Edinburgh

Cheng, G., Yang, C., Yao, X., Guo, L., Han, J. (2018) When Deep Learning Meets Metric Learning: Remote Sensing Image Scene Classification via Learning Discriminative CNNs. *IEEE Transactions on Geoscience and Remote Sensing*. 56(5), pp. 2811-2821. doi: 10.1109/TGRS.2017.2783902

Chollet, Francois (2015) Keras. Available at: https://github.com/fchollet/keras

Engstrom, R., Hersh, J. and Newhouse, D. (2017). Poverty from space: using high-resolution satellite imagery for estimating economic well-being*. Washington, D.C.: World Bank Group.* Available at: http://documents.worldbank.org/curated/en/610771513691888412/Poverty-from-space-using-high-resolution-satellite-imagery-for-estimating-economic-well-being [Accessed 23 Jan. 2020]

Esri. ArcMap: An overview of the Segmentation and Classification toolset. Available at: https://desktop.arcgis.com/en/arcmap/10.3/tools/spatial-analyst-toolbox/an-overview-of-the-segmentation-and-classification-tools.htm [Accessed: 12th of Mar, 2020]

Etten, Adam Van (2018). You Only Look Twice: Rapid Multi-Scale Object Detection In Satellite Imagery. *Submitted to Knowledge Discovery in Databases competition,* arXiv:1805.09512

Everingham, M., Eslami, S.M.A., Van Gool, L., Williams, C.K.I., Winn, J., Zisserman, A. (2015) The Pascal Visual Object Classes Challenge: A Retrospective. *International Journal of Computer Vision*. 111(1) pp. 98–136

Foody, G.M. (2002) Status of land cover classification accuracy assessment. *Remote Sensing of Environment*. 80(1) pp. 185–201

Guigoz, Y., Giuliani, G., Nonguierma, A., Lehmann, A., Mlisa, A., Ray, N. (2017). Spatial Data Infrastructures in Africa: A Gap Analysis. *Journal of Environmental Informatics* 30(1) pp. 53-62. DOI:10.3808/jei.201500325

Gulli, A. & Pal, S. (2017) *Deep learning with Keras*, Packt Publishing Ltd

Guo, Wei, Wen Yang, Haijian Zhang, and Guang Hua (2018). Geospatial Object Detection in High Resolution Satellite Images Based on Multi-scale Convolutional Neural Network. *Remote Sensing* 10(1).

International Fund for Agricultural Development (IFAD) Annual Report (2019). Rome, Italy. www.ifad.org/ar2019

Jean, N., Burke, M., Xie, M., Davis, W.M., Lobell, D.B., Ermon, S. (2016) Combining satellite imagery and machine learning to predict poverty. *Science*. 353 (6301) pp. 790–794. Available from: https://dx.doi.org/10.1126/science.aaf7894

Keras (2018) Keras - High-level neural networks API. https://keras.io/

Längkvist, Martin, Kiselev, Andrey, Alirezaie, Marjan, and Loutfi, Amy (2016) Classification and Segmentation of Satellite Orthoimagery Using Convolutional Neural Networks. *Remote Sensing* 8:4 pp.329.

Li, J., Huang, X., Gong, J. (2019) Deep neural network for remote-sensing image interpretation: status and perspectives. *National Science Review*. 6 (6) pp. 1082–1086

Jia Song, Shaohua Gao, Yunqiang Zhu & Chenyan Ma (2019) A survey of remote sensing image classification based on CNNs. *Big Earth Data.* 3(3), pp.232-254, DOI: 10.1080/20964471.2019.1657720

Jozdani, Shahab Eddin, Brian Alan Johnson, and Dongmei Chen (2019) Comparing Deep Neural Networks, Ensemble Classifiers, And Support Vector Machine Algorithms For Object-Based Urban Land Use/Land Cover Classification. *Remote Sensing* 11(14) pp.1713-1730. DOI:10.3390/rs11141713

Map Library (Africa) (2007). *Map Maker Ltd.* Available at: http://www.maplibrary.org/library/stacks/Africa/index.htm [Accessed: 9th of July, 2020]

Maggiori, E., Tarabalka, Y., Charpiat, G., Alliez, P. (2017) Convolutional Neural Networks for Large-Scale Remote-Sensing Image Classification. *IEEE Transactions on Geoscience and Remote Sensing.* 55 (2) pp. 645–657. Available from: https://dx.doi.org/10.1109/TGRS.2016.2612821

Michalitsianos, Gerasimos A. (2019). command-line program for multispectral pan-sharpening (Python). Includes Brovey, FIHS (Fast Intensity Hue Saturation), Wavelet, and PCA (Principal

Component Analysis) *GitHub*. Available at: https://github.com/gerasimosmichalitsianos/pansharpen [Accessed: 2nd of Mar, 2020]

Mitchell, S., Gelman, A., Ross, R., et al. (2018) The Millennium Villages Project: a retrospective, observational, endline evaluation. *The Lancet Global Health*. 6(5) pp. e500–e513

Mylonas, S.K.; Stavrakoudis, D.G.; Theocharis, J.B.; Mastorocostas, P.A. (2015) A region-based genesis segmentation algorithm for the classification of remotely sensed images*. Journal of Remote Sensing*. 7(1) pp.2474–2508

Nguyen, G., Dlugolinsky, S., Bobák, M., Tran, V., Garcia, L.A., Heredia, I., Malik, P., Hluchy, L. (2019) Machine Learning and Deep Learning frameworks and libraries for large-scale data mining: a survey. *Artificial Intelligence Review* [online]. 52 (1) pp. 77–124. Available from: https://dx.doi.org/10.1007/s10462-018-09679-z

Okwi, P.O., Ndeng'E, G., Kristjanson, P., et al. (2007) Spatial determinants of poverty in rural Kenya. Proceedings of the National Academy of Sciences [online]. 104 (43) pp. 16769–16774. Available from: https://dx.doi.org/10.1073/pnas.0611107104

Penatti, O.A., Nogueira, K., dos Santos, J.A. (2015) Do deep features generalize from everyday objects to remote sensing and aerial scenes domains? In Proceedings of *the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, Boston, MA, USA, 7–12 June, 2015

Radoux, J. and Bogaert, P.,(2017) Good Practices for Object-Based Accuracy Assessment. *Remote Sensing.* 9(7), pp 646. https://doi.org/10.3390/rs9070646

Ritwik, G., Hosfelt, R., Sajeev, S., Patel, N., Goodman, B., Doshi, J., Heim, E., Choset, H., Gaston, M. (2019) A Dataset for Assessing Building Damage from Satellite Imagery. *xView2* Available at: https://xview2.org/dataset [Accessed: 12th Mar, 2020]

Ruscica, Tim (2020). TensorFlow 2.0 Complete Course - Python Neural Networks for Beginners. *FreeCodeCamp.org.* Available at: https://www.freecodecamp.org/news/massive-tensorflow-2-0-free-course/ [Accessed: 9th of Mar, 2020]

Salcedo-Sanz, S., Ghamisi, P., Piles, M., Werner, M., Cuadra, L. Moreno-Martinez, A., Izquierdo-Verdiguier, E., Munoz-Mari, J., Mosavi, A., Camps-Valls, G. (2020) Machine learning information fusion in Earth observation: A comprehensive review of methods, applications and data sources. *Information Fusion*. 63 pp. 256–272

Shorten, C., Khoshgoftaar, T.M. (2019) A survey on Image Data Augmentation for Deep Learning. *J Big Data* 6(60). https://doi.org/10.1186/s40537-019-0197-0

Steele, J.E., Sundsøy, P.R., Pezzulo, C., Alegana, V.A., Bird, T.J., Blumenstock, J., Bjelland, J., Engø-Monen, K., de Montjoye, Y., Iqbal, A.M., Hadiuzzaman, K.N., Lu, X., Wetter, E., Tatem, A.J., Bengtsson, L. (2017) Mapping poverty using mobile phone and satellite data. *Journal of The Royal Society Interface.* 14(127) pp. 1742-5689. Available from: https://dx.doi.org/10.1098/rsif.2016.0690

TensorFlow Core v2.3.0. tf.keras.callbacks.EarlyStopping. *Google Brain.* Available at: https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/EarlyStopping

Trimble Inc. (2019). Trimble eCognition Suite. [Online]. Available at: https://docs.ecognition.com/v9.5.0/Page%20collection/eCognition%20Suite%20Dev%20UG.htm. Accessed on: July 10, 2020

Tripathy, Pratyush (2019). Neural Network for Satellite Data Classification Using Tensorflow in Python. A step-by-step guide for Landsat 5 multispectral data classification. *Towards data Science.* Available at: https://towardsdatascience.com/neural-network-for-satellite-data-classification-using-tensorflow-in-python-a13bcf38f3e1 [Accessed: 11th of Mar, 2020]

UN (2020), The Sustainable Development Goals Report 2020, UN, New York, https://unstats.un.org/sdgs/report/2020/The-Sustainable-Development-Goals-Report-2020.pdf

UN (2019), End Poverty in all Its Forms Everywhere 2019, UN, New York, https://unstats.un.org/sdgs/report/2019/goal-01/

United Nations Development Program (UNDP)(2007) Millennium Villages Project. Available at: http://mv-aid.org/mv/mbola.html

Vafeiadis, T., Diamantaras,K.I., Sarigiannidis, G., Chatzisavvas, K.C. (2015) A comparison of machine learning techniques for customer churn prediction. *Simulation Modelling Practice and Theory*, 55 pp. 1-9

Watmough, G., Marcinko, C., Sullivan, C., Tschirhart, K., Mutuo, P., Palm, C. and Svenning, J. (2019) Socioecologically informed use of remote sensing data to predict rural household poverty. *Proceedings of the National Academy of Sciences*, 116:4, pp.1213-1218

World Bank (2015) PovcalNet online poverty analysis tool. Available at: http://iresearch.worldbank.org/povcalnet/

Schweitzer, D. and Agrawal, R. (2018) Multi-Class Object Detection from Aerial Images Using Mask R-CNN. *IEEE International Conference on Big Data*, Seattle, WA, USA, pp. 3470-3477

Zhang, C., Bengio, S., Hardt, M., Recht, B,. Vinyals, O. (2017). Understanding deep learning requires rethinking generalization. *ICLR.* arXiv:1611.03530

Part II: Technical Report

Table of Contents

1. List of Figures and Tables

2. Introduction

This document is a supplementary resource for the research paper and includes an extended overview of Convolutional Neural Networks (CNNs), a more detailed methodology, as well as possible solutions for improvement of the overall algorithm.

The aim of this technical report is to allow readers to gain a fuller understanding of the methods and tools that went into developing this algorithm. Any user should be able to replicate and improve upon the current methodology for further research purposes.

3. An Overview of Convolutional Neural Networks and the Training Process

Convolutional Neural Networks (CNNs) are a type of machine learning (ML) that learn patterns within image data by extracting localized features from that data using a series of linear algebra computations.

The function of a CNN is to learn patterns from a set of training images. Once the CNN has 'trained' sufficiently, it can be used to recognize the same patterns in data it hasn't seen before.

The training process works by using a moving window of specified size to extract low-level features (edges and corners) and performs a dot-product on each window's values with a set of random weights and biases. This produces a stack of 'feature' maps for each layer of input data, (R,G,B, and NIR in the case of multispectral data). This process is called a convolution and multiple convolutions can be combined by using the output of one as the input of another. The more convolutions in sequence, the higher level of features extracted (Gulli & Pal, 2017).

Further calculations are performed to simplify and smooth data between each convolution including normalization and maximum pooling. Normalization is accomplished using activation functions, of which there are several types. Common activations functions are Rectified Linear Unit (ReLu) and hyperbolic tangent functions. These functions usually make sure all resulting values are positive and between 0 and 1, depending on the data and the activation function being used. Maximum pooling is a way of down sampling data by taking the maximum of non-overlapping regions of a specific size. Pooling size is often 2x2 pixels, which allows slight rotational and translational variations to be learned by the model which can increase the generalization ability of the model (Längkvist, et al., 2016).

The final step is flattening these output vectors into a one-dimensional vector called the 'fully connected layer'. This layer outputs a set of predictions for the training data given to it (TensorFlow Core).

As the data moves through these convolutions and calculations to produce prediction values, this is called a 'forward pass' or an 'epoch'. The output predictions are then used to calculate the 'loss' of that epoch by comparing them to the true labels. Loss can be calculated in many different ways by using different kinds of loss functions, including Mean Squared Error (MSE), Intersection Over Union (IOU) for bounding box information, and many other more complicated equations. This information is then passed back through the model in a process called back-propagation to adjust weights and biases (TensorFlow Core).

In order for these weights and biases to be nudged in the right direction, an 'optimizer function' is used to conduct a process called 'gradient-descent'. This is a mathematical function, which finds local minima by using the relative changes (which can be thought of as slopes/gradients) from previous epochs. Different optimizer functions with adjustable parameters such as learning rate and decay are used for optimization of the training process (TensorFlow Core).

The process of gradient descent can be visualized as a 3D topographical map. For instance, if the learning rate is too high, the gradient descent function can get stuck bouncing back and forth between two 'valley' walls and never reach the minimum of the valley (which represents the lowest possible loss and highest possible accuracy). However, if the learning rate is too low, the function can get stuck in a local minima, which may not represent the global minimum (lowest topographical point on the map) (Bottou, 2010).

Training should be stopped once the gradient descent has reached its conclusion and loss has stopped significantly decreasing (TensorFlow Core).

4. Comparison of Pansharpening Techniques

Four different pansharpening techniques were tested on a clipped subset of the Mbola multispectral and panchromatic data using a python script from Michalitsianos, 2019 (Appendix I). These techniques included the Brovey, Principle Component Analysis (PCA), Fast Intensity Hue Saturation (FIHS), and Wavelet techniques (Figure 1).

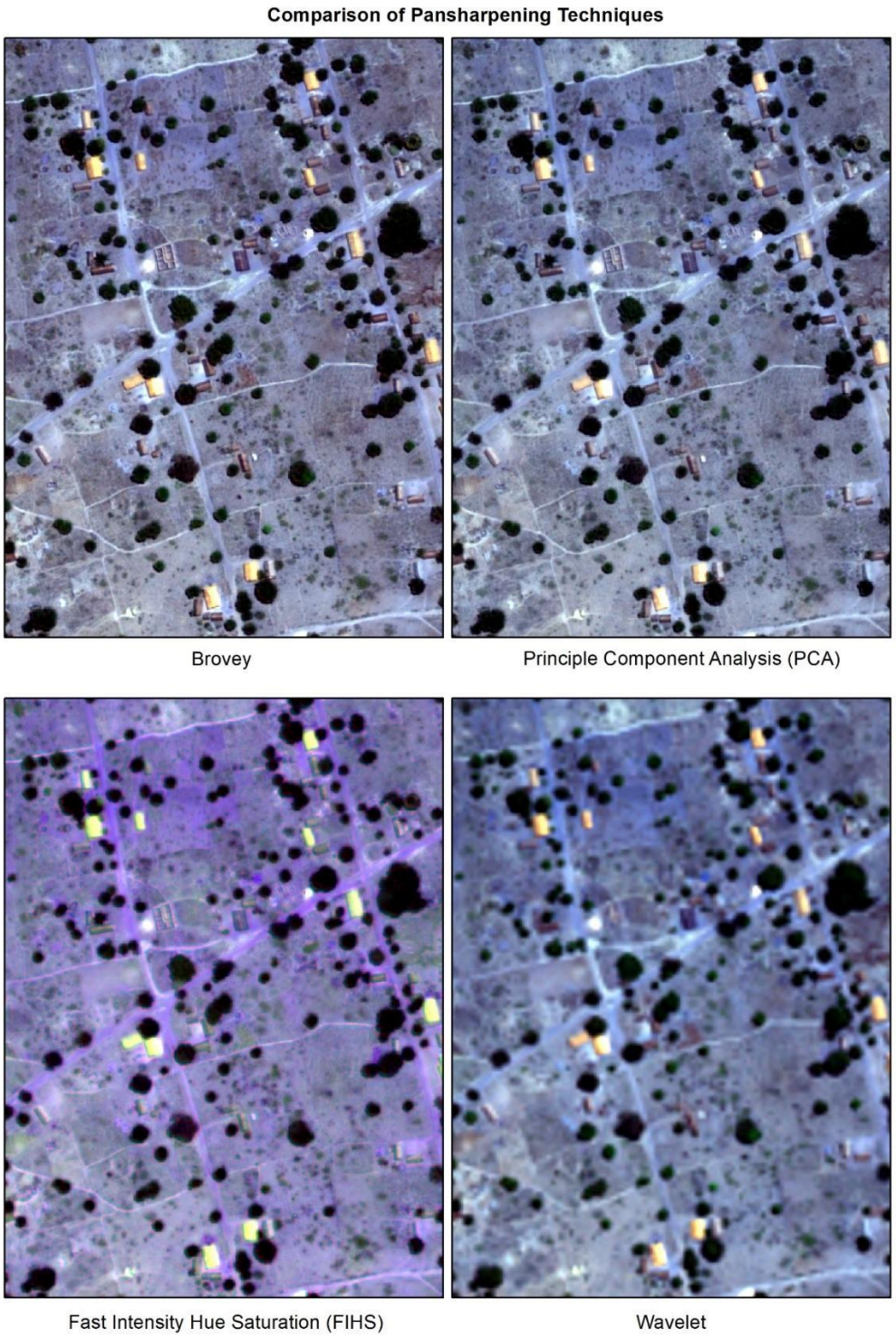


Figure 1: Comparison of pansharpened results using four different techniques shown in true color

The Brovey and PCA techniques appeared to produce the best results with the highest spatial resolution and the least hue distortion. The Brovey technique was chosen as it appears to produce a slightly sharper and less hue-affected output and was more computationally efficient.

5. Steps for Creating Training Data

The ArcMap tools and workflow for creating training data can be seen in Figure 2.

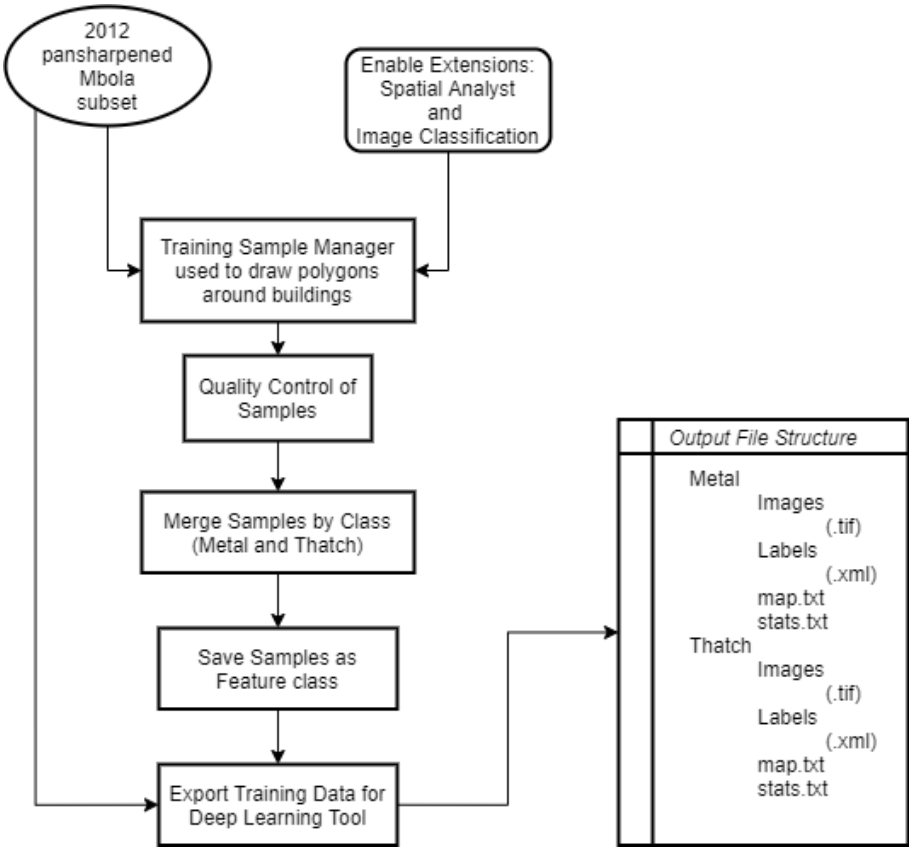


Figure 2: Workflow for creating training data in ArcMap.

The ruleset for inclusion and exclusion of samples from the training dataset is as follows: Potential buildings smaller than four square meters (16 pixels) were excluded from training data as this scale is more likely to be descriptive of outhouses or vehicles. When digitizing buildings above this size threshold, obscuring trees and areas of significant pixel bleeding were excluded. Building shadows were included, as this is often a good distinguishing characteristic between roofs and ground features with similar spectral signatures, such as bright fields (Schweitzer & Agrawal, 2018).

The parameters used for the 'Export Training Data For Deep Learning Tool' can be seen in Figure 3.

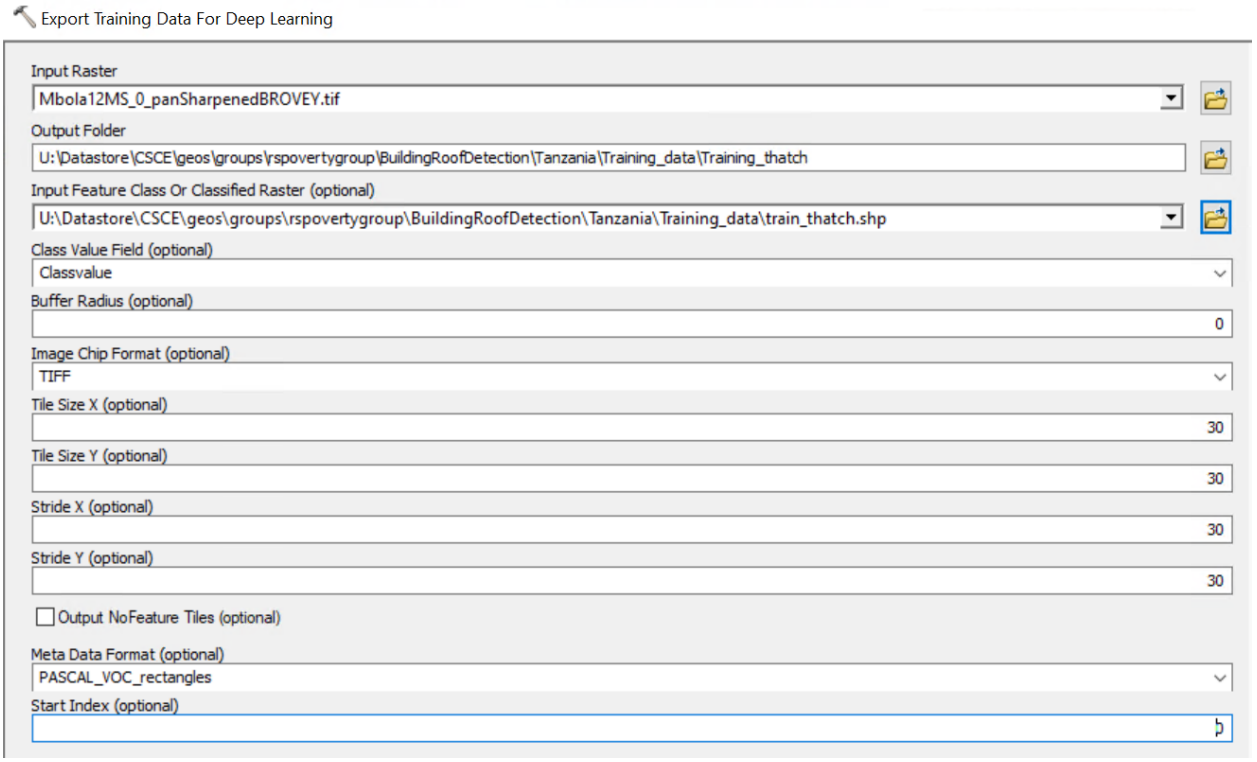


Figure 3: Screen shot of parameters used for the Export Training Data for Deep Learning tool in ArcMap.

This tool was set to export training samples as tiffs for compatibility with all other data. Tile size was set to 30x30 pixels based on the average size of buildings in the dataset and for ease of processing within the CNN training process. The stride size was also set to 30x30 to ensure no overlap.

The same tool was run with 'Output NoFeature Tiles' to produce background samples of the same size and format. The Background samples were evaluated visually in ArcMap to make sure no roofs were present since not all roofs in the image were selected as training samples. Python was used to convert this data into numpy arrays for input into a CNN model (see Appendix I).

The output format of data was set to PASCAL_VOC, which allows for use in an object detection model and also for easy manipulation in Python. Start index is mostly used for batch processing when using the tool more than once on the same dataset. Since this was not needed, the start index was kept at the default value, zero.

6. Justification for Chosen Hyperparameters

The final chosen hyperparameters can be seen in Table 1.

Table 1: List of final architectures and hyperparameters used for the training process (see Appendix II; Appendix III).

Hyperparameters	*Classifier CNN*	*Object Detection CNN*
Convolutions	3	3
Activation Functions	ReLu and Softmax	Leaky ReLu and Linear
Optimizer	Adam	Adam
Loss Function	Sparse Categorical Crossentropy	Custom
Input shape	(Num Training Samples,30,30,4)	(Num Training Samples, 30,30,4)
Output shape	(Num Training Samples, 3)	(Num Training Samples, 7)

The number of convolutions was kept low in an effort to avoid 'over-fitting' and due to the limited size of training data, as well as the low level features associated with roof shape (Simonyan et al., 2015;
Song et al., 2019).

The activation function was set to the default Keras value of Rectified Linear Unite (ReLu), which makes sure values throughout the training process are non-negative and has had success in many CNN applications. Similarly, the optimizer was set to 'Adam', a commonly successful parameter (Keras; Zhu et al., 2017).

The input shape was set to be compatible with amount and size of input training samples and output shape (or prediction shape) was set to be compatible with the amount of training data and number of classification labels (No Roof, Metal, and Thatch).

```
# Build simple classifier model from scratch (3 Convolutions)
img_size = 30

model_3Convs = Sequential()
model_3Convs.add(Conv2D(32, kernel_size=(3, 3), strides=(1,1),
                 activation='relu',
                 input_shape=(img_size,img_size,4)))
model_3Convs.add(Conv2D(64, (3, 3), activation='relu'))
model_3Convs.add(Conv2D(128, (3, 3), activation='relu'))
model_3Convs.add(GlobalMaxPooling2D())
model_3Convs.add(Dropout(0.25))
model_3Convs.add(Dense(256, activation='relu'))
model_3Convs.add(Dropout(0.5))
model_3Convs.add(Dense(3, activation='softmax'))

print(model_3Convs.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 32)        1184
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 26, 26, 64)        18496
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 24, 24, 128)       73856
_________________________________________________________________
global_max_pooling2d_1 (Glob (None, 128)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               33024
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 771
=================================================================
Total params: 127,331
Trainable params: 127,331
Non-trainable params: 0
```

Figure 4: Final code for building a simple convolutional neural network in Google Collaboratory and the resulting architecture.

7. Attempting to Create an Object Detection Model

In order to convert the CNN Classifier into an Object Detection model, several manipulations of the CNN hyperparameters and training code were required (Table 1; Appendix III). The basic architecture, with three convolutional layers, window size and number of filters were kept the same. The optimizer was unchanged. The activation functions were set to Leaky ReLu and Linear for the Dense layer. These are both commonly used parameters in object detection CNN's (Keras).

The main changes were the creation of a custom loss function and the input shape, which was set to receive bounding box information as well as one-hot encoded label information. One-hot-encoded labels are simply labels as vectors with placeholders for each class and values of 0 or 1 representing a class. For example, if:

No Roof = 0, Metal = 1, Thatch = 2

and a sample is labeled as 2, the one-hot encoded label would = [0,0,1]. Similarly, 1 = [0,1,0] and 0 = [1,0,0].

The custom loss function was based on the general methodology of recently-developed object detection model, You Only Look Once (YOLO) (Redmon et al., 2015). It involved combining calculations for location error metrics such as Intersection Over Union (IOU), and classification error metrics such as Mean Square Error (MSE) and sparse categorical crossentropy. Modifications also had to be made to the training code to gain a higher level of control than the Keras API generally allows (Chollet, 2015). This was done using gradient tape and custom forward pass functions (Keras).

The training process took much longer than that of the Classifier and was unable to complete. The final accuracy was up 7% before training had to be stopped. It is unknown whether this model would have been successful or not.

8. Utilizing Probability Thresholds

The raw outputs of the CNN Classifier model were probability vectors where the first value is the probability that the input sample is No Roof, the second value is the probability that the input sample is Metal, and the third value is the probability that the input sample is Thatch. In order to simplify the classification process and make it uniform across data subsets, the index of the maximum probability was chosen as the classification for that sample. For example, take the following example of an output probability vector:

[0.001, 0.209, 0.700]

The classification of that sample would be Thatch because the placeholder for thatch shows the highest probability, or a 70% chance that the sample contains a thatch roof. However, one way to improve the model is by searching for patterns in the relationships between probabilities and setting thresholds based on those relationships.

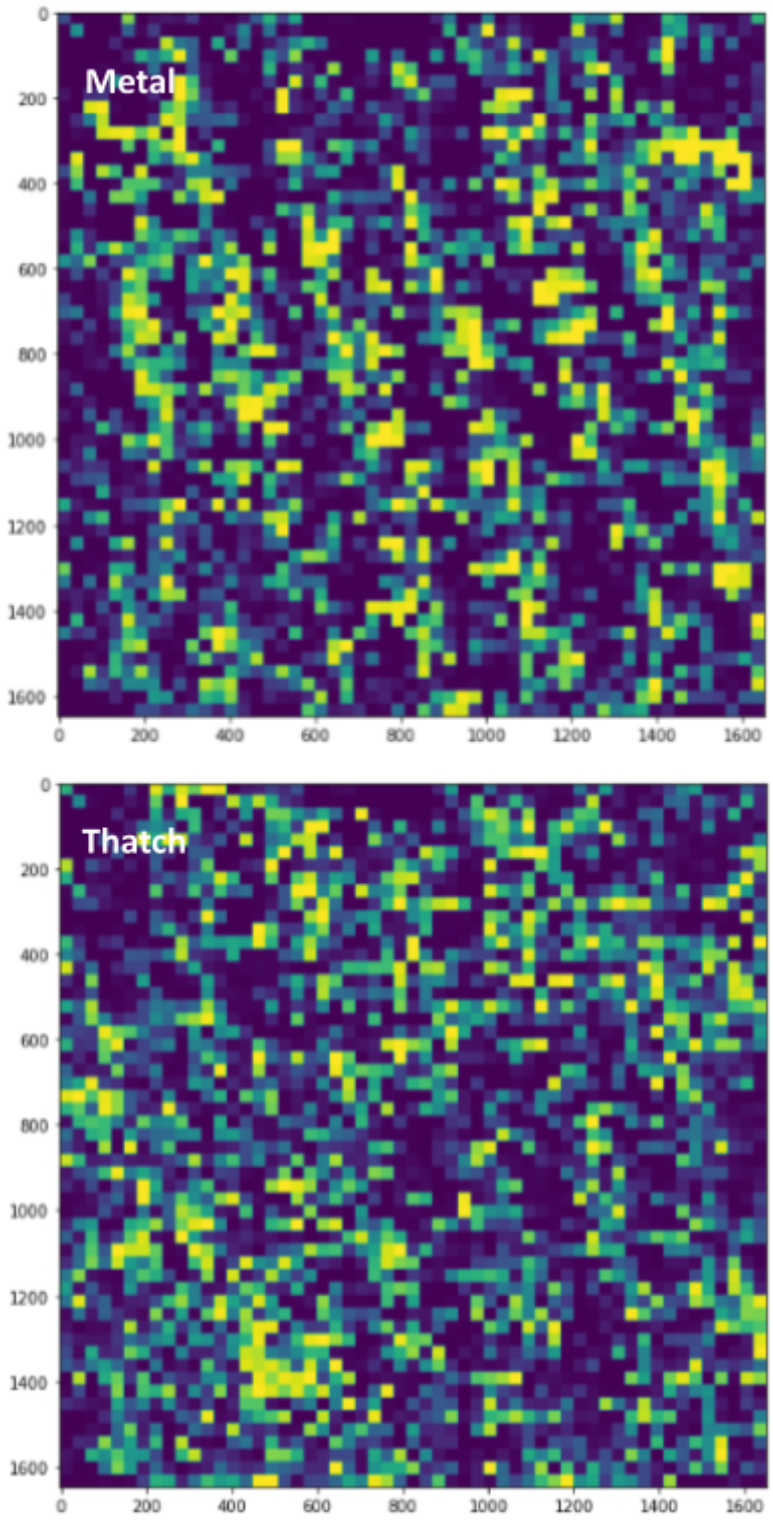


Figure 5: Example outputs of 3-band probability masks produced by final CNN model for 2012 Village 1.

9. Segmentation

Two separate segmentation workflows were tested in ArcMap. The first involved using the Multivariate toolbox to include all four color bands. The workflow for this segmentation included the following tools in sequence:

ISO Cluster > Maximum Likelihood Classification > Reclassify Confidence output to (0-13=No Roof, 14=Roof).

However, this process provides few options for user-control over the segmentation (largely unsupervised), and only seemed to work well for classifying 2012 metal roofs.

The second segmentation workflow involved using the Segmentation and Classification toolbox, which only allows three color bands as input. This workflow can be seen in Figure 6.

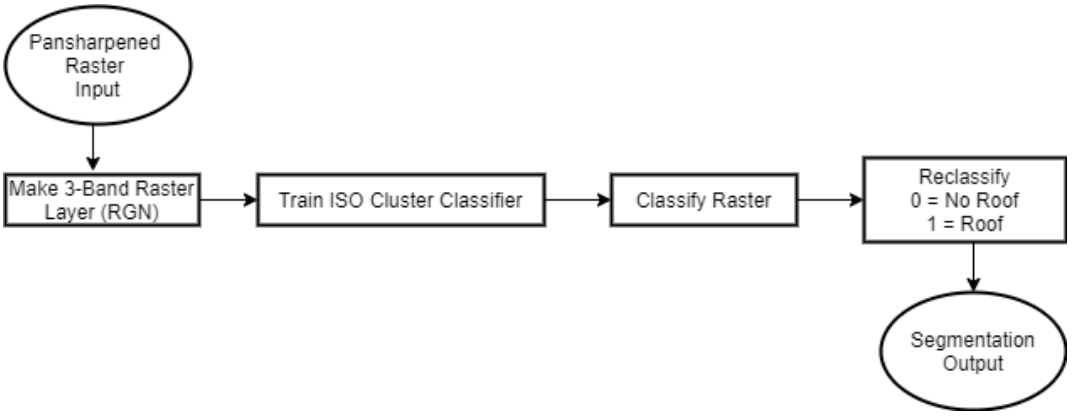


Figure 6: Final segmentation workflow conducted in ArcMap to be combined with CNN Classifier output label masks.

Two different color band combinations (RGN and RBN) were tested using this workflow to see which produced the best segmentations. The RBN (Red, Blue, and NIR) combination seemed to work best for most of the datasets and was therefore chosen as input for the final model.

The 3-band segmentation process allowed for more supervision of the ruleset, including choosing parameters such as Rectangularity and Compactness for consideration in segmenting images. In the 'Train ISO Cluster Classifier' tool, a large number of classes equal to 20 was selected for segmentation. This was done to further be able to supervise the segmentation process and potentially choose multiple classes for recombination later on. Changing the maximum number of iterations (a range of values were tested from 1-100) did not seem to have any effect on the output. Therefore, the final iteration value was set to 20, which is on the high end of what ESRI recommends for most datasets (ESRI, 2019). Color, Mean, Rectangularity and Compactness were chosen as parameters for segmentation consideration.

This workflow, while still missing some functionality and supervision that eCognition provides, was much more effective across datasets than the Multivariate segmentation workflow, especially for thatch roofs.

10. Post-Processing Steps

All post-processing was done in ArcMap using toolbox tools and arcpy. The Create Feature Class tool was used to manually create ground truth data for four of the largest villages in Mbola. Raster Calculator was used to combine CNN Classifier Masks and Segmentation outputs. An arcpy script was used for all other post-processing steps (see Appendix VI). This arcpy script was run for each dataset, a total of 18 times:

(2 Roof Classes * 4 Villages * 2 Years) + Zimbabwe Metal + Zimbabwe Thatch

The total Post-Processing workflow can be seen in Figure 7. This involved combining and transforming the model outputs into polygon building predictions and using the Intersect tool to compare these predictions to ground truth polygons.

For polygon conversion, shapes were preserved by specifying 'No simplify' in the arcpy script. For the Aggregate Polygons tool, a distance threshold of 4 meters was specified, a minimum size threshold of 2 square meters was specified, and 'NON_ORTHOGONAL' was specified in order to preserve original shape and not lose information (ESRI, 2016). All other values were set to defaults.

The Intersect tool was used twice for each dataset, once to compare predictions with ground truth polygons of the same class to produce True Positives, and once to compare with ground truth polygons of the opposite class to produce Misclassified roof predictions. These values were then plugged into a confusion matrix and subtracted from the total number of predictions and true roofs for each class to produce False Negative and False Positive values.

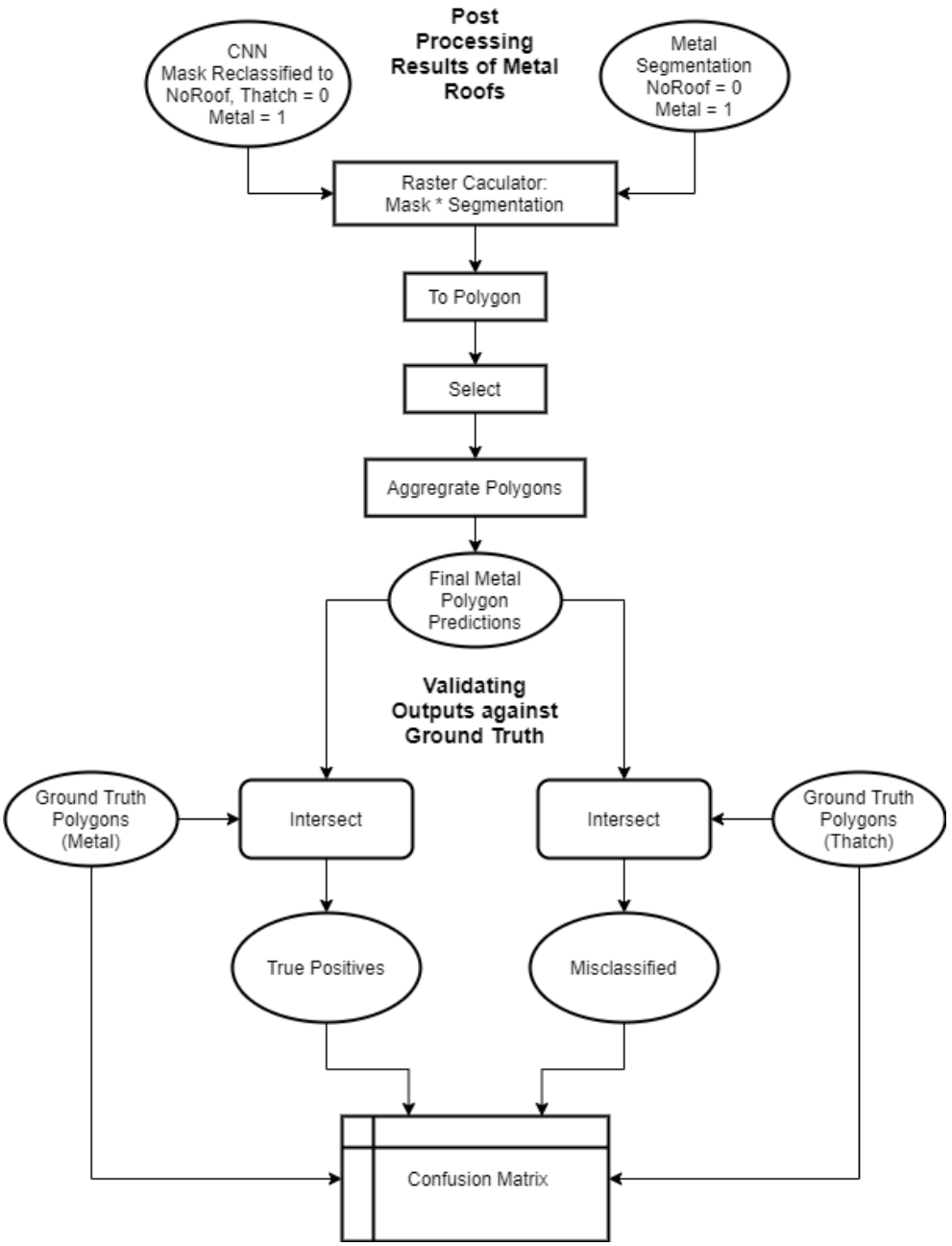


Figure 7: Post-processing and validation workflow for a metal dataset conducted in ArcMap. The same workflow structure applies for thatch (but with Thatch replacing Metal and vice versa).

11. References for Technical Report

Abadi, A., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jozefowicz, G., Jia, Y., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Schuster, M., Monga, R., Moore, S., Murray, D., Olah, D., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X. (2015) TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from: tensorflow.org

Bottou, L. (2010) Large-Scale Machine Learning with Stochastic Gradient Descent. *NEC Labs America, Princeton, NJ*. pp. 177–186. Available from: https://dx.doi.org/10.1007/978-3-7908-2604-3_16

Brownlee, Jason. (2019). A Gentle Introduction to Object Recognition With Deep Learning. *Deep Learning For Computer Vision.* Available at: https://machinelearningmastery.com/object-recognition-with-deep-learning/

Chollet, Francois (2015) Keras. Available at: https://github.com/fchollet/keras

ESRI (2016). ArcMap: An overview of the Cartography toolbox; Aggregate Polygons. Available at: https://desktop.arcgis.com/en/arcmap/10.3/tools/cartography-toolbox/aggregate-polygons.htm [Accessed: 30th of July, 2020].

Gulli, A. & Pal, S. (2017) *Deep learning with Keras*, Packt Publishing Ltd

Jozdani, Shahab Eddin, Brian Alan Johnson, and Dongmei Chen (2019) Comparing Deep Neural Networks, Ensemble Classifiers, And Support Vector Machine Algorithms For Object-Based Urban Land Use/Land Cover Classification. *Remote Sensing* 11(14) pp.1713-1730. DOI:10.3390/rs11141713

Keras. Keras API reference: Layers API: Layer Activation Functions. Available at: https://keras.io/api/layers/activations/

Längkvist, Martin, Kiselev, Andrey, Alirezaie, Marjan, and Loutfi, Amy (2016) Classification and Segmentation of Satellite Orthoimagery Using Convolutional Neural Networks. *Remote Sensing* 8:4 pp.329.

Li, Z., Li, Y. S., Wu, X., Liu, G., Lu, H., & Tang, M. (2017) Hollow village building detection method using high resolution remote sensing image based on CNN. *Transactions of the Chinese Society of Agricultural Machinery*. 48(1) pp.160–165

Nguyen, G., Dlugolinsky, S., Bobák, M., Tran, V., Garcia, L.A., Heredia, I., Malik, P., Hluchy, L. (2019) Machine Learning and Deep Learning frameworks and libraries for large-scale data mining: a survey. *Artificial Intelligence Review* [online]. 52 (1) pp. 77–124. Available from: https://dx.doi.org/10.1007/s10462-018-09679-z

Redmon, J., Divvala, S., Girshick, R., Farhadi, A. (2015) You Only Look Once: Unified, Real-Time Object Detection. *Computer Vision and Pattern Recognition; Cornell University.* arXiv:1506.02640

Ruscica, Tim (2020). TensorFlow 2.0 Complete Course - Python Neural Networks for Beginners. *FreeCodeCamp.org.* Available at: https://www.freecodecamp.org/news/massive-tensorflow-2-0-free-course/ [Accessed: 9th of Mar, 2020]

Salcedo-Sanz, S., Ghamisi, P., Piles, M., Werner, M., Cuadra, L. Moreno-Martinez, A., Izquierdo-Verdiguier, E., Munoz-Mari, J., Mosavi, A., Camps-Valls, G. (2020) Machine learning

information fusion in Earth observation: A comprehensive review of methods, applications and data sources. *Information Fusion*. 63 pp. 256–272

Schweitzer, D. and Agrawal, R. (2018) Multi-Class Object Detection from Aerial Images Using Mask R-CNN. *IEEE International Conference on Big Data*, Seattle, WA, USA, pp. 3470-3477

Simonyan, K., Zisserman, A. (2015) Very Deep Convolutional Networks for Large-Scale Image Recognition. *Visual Geometry Group, Department of Engineering Science, University of Oxford.* Published as conference paper ICLR. arXiv:1409.1556

Shorten, C., Khoshgoftaar, T.M. (2019) A survey on Image Data Augmentation for Deep Learning. *J Big Data* 6(60). https://doi.org/10.1186/s40537-019-0197-0

TensorFlow Core. Tutorials. https://www.tensorflow.org/tutorials

Zhang, C., Bengio, S., Hardt, M., Recht, B,. Vinyals, O. (2017). Understanding deep learning requires rethinking generalization. *ICLR.* arXiv:1611.03530

Zhu, X.X., Tuia, D., Mou, L., Xia, L.Z., Xu, F., Fraundorfer, F. (2017) Deep Learning in Remote Sensing: A Comprehensive Review and List of Resources. *IEEE Geoscience and Remote Sensing Magazine*. 5 (4) pp. 8–36

12. Appendix

I: Preprocessing Script

```
# PREPROCESSING.ipynb
'''This script is primarily for the preprocessing of data for input into the Keras
CNN
models. More thatch training data and more augmentation (histogram stretch) techniq
ues
were recently added.'''

# Imports
import os
import copy
import h5py
import numpy as np
from osgeo import gdal
import matplotlib.pyplot as plt
import matplotlib.patches as patches
%matplotlib inline

# Paths to data and other global variables
# Thatch (extracted from multiple tiffs in order to obtain enough samples)
thatch_root0 = "/content/drive/My Drive/UoE_GIS/Training_thatch/"
thatch_map0 = "/content/drive/My Drive/UoE_GIS/Training_thatch/map.txt"
thatch_root2 = "/content/drive/My Drive/UoE_GIS/Training_thatch_M2/"
thatch_map2 = "/content/drive/My Drive/UoE_GIS/Training_thatch_M2/map.txt"
thatch_root3 = "/content/drive/My Drive/UoE_GIS/Training_thatch_M3/"
thatch_map3 = "/content/drive/My Drive/UoE_GIS/Training_thatch_M3/map.txt"
thatch_root4 = "/content/drive/My Drive/UoE_GIS/Training_thatch_M6/"
thatch_map4 = "/content/drive/My Drive/UoE_GIS/Training_thatch_M6/map.txt"
# Metal
metal_root = "/content/drive/My Drive/UoE_GIS/Training_metal/"
metal_map = "/content/drive/My Drive/UoE_GIS/Training_metal/map.txt"
# No roof / Background
noroof_root = "/content/drive/My Drive/UoE_GIS/Training_noroof/images/"

# Training sample size (num pixels in x and y directions)
img_size = 30

# Function to generate lists of tiff files and label files
def createTifflists(root, map):
  tiffList = []
  labelList = []
  mapfile = open(map, "r")
  lines = mapfile.readlines()
  for l in lines:
    l = l.replace('\\', '/')        # replace backslashes with forward slashes
    l = l.split(' ')                # split line by spaces
    tiffList.append(root + l[0])    # append tif filename to tiff list
    labelList.append(root + l[2])   # append xml filename to label list
  mapfile.close()
  return (tiffList, labelList)

# Call above function for thatch and metal directories
thatch_tiffList0, thatch_labelList0 = createTifflists(thatch_root0, thatch_map0)
thatch_tiffList2, thatch_labelList2 = createTifflists(thatch_root2, thatch_map2)
thatch_tiffList3, thatch_labelList3 = createTifflists(thatch_root3, thatch_map3)
thatch_tiffList4, thatch_labelList4 = createTifflists(thatch_root4, thatch_map4)
metal_tiffList, metal_labelList = createTifflists(metal_root, metal_map)

# Combine thatch directories
```

```
thatch_tiffList = thatch_tiffList0 + thatch_tiffList2 + thatch_tiffList3 + thatch_t
iffList4
thatch_labelList = thatch_labelList0 + thatch_labelList2 + thatch_labelList3 + that
ch_labelList4

# No roof / Background
noroof_tiffList = []
fileList = os.listdir(noroof_root)
for file in fileList:
    if file.endswith('.tif'):
        filename = noroof_root + file
        noroof_tiffList.append(filename)


# Sort noroof_tiffList by file number and get rid of 'bad' samples that contain roo
fs #(evaluated in ArcMap)
print ('Original length of noroof_tiffList:', len(noroof_tiffList))
rmtiff = []
getnum = []
badList = ['00095','00098','00921','00924','00677','00680','00965','00968','01745',
           '01747','01748','01750','01751','01754','01789','01790','01791','01792',
           '01794','02569','02571','02572','02573','02574','02575','02576','02577',
           '02578','02580','02611','02613','02614','02615','02616','02617','02618',
           '02620','02570','02567','02609','02612','02619','02621','02622','02624',
           '03397','03400','03401','03404','03433','03434','03435','03436','03437',
           '03438','03439','03440','03441','03442','03443','03444','03445','03446',
           '03448','03451','03454','04255','04257','04258','04259','04260','04262',
           '04263','04266','04267','04270','04275','04277','04278','04280']

for tiff in noroof_tiffList:
  rmtiff.append(tiff.strip('.tif'))     # Strip '.tif's

for i in rmtiff:
  getnum.append(i[-5:])                 # Put last five characters into getnum list
getnum = sorted(getnum)                 # for numerical sorting

for i in range(0, len(noroof_tiffList)):
  noroof_tiffList = sorted(noroof_tiffList, key = lambda getnum: getnum)    # Sort
based on getnum

for bad in badList:
  bad = (noroof_tiffList[0][:-9] + bad + '.tif')  # Add whole path elements to badL
ist
  if bad in noroof_tiffList:                      # search for and remove bad tiffs
    noroof_tiffList.remove(bad)                   # from noroof_tiffList

print ('Sorted and pruned noroof_tiffList: ', len(noroof_tiffList))

# Function to convert tiffs into numpy arrays using gdal
def tiff_toArray(tiffList):
  data_size = len(tiffList)
  # Create array of zeros with proper shape
  data = np.zeros([data_size,img_size,img_size,4], dtype=float)
  count = 0     # To keep track of tiff index

  for tiff in tiffList:
      ds=gdal.Open(tiff)

      # read data. Returns as a 2D numpy array
      NIR=ds.GetRasterBand(4).ReadAsArray()
      blue=ds.GetRasterBand(3).ReadAsArray()
      green=ds.GetRasterBand(2).ReadAsArray()
      red=ds.GetRasterBand(1).ReadAsArray()
```

```
        # Insert band values into proper place in array
        data[count,:,:,3] = NIR
        data[count,:,:,2] = blue
        data[count,:,:,1] = green
        data[count,:,:,0] = red
        count += 1
    return (data)

# Run tiff_toArray function on different class lists (thatch, metal, noroof)
dataThatch = tiff_toArray(thatch_tiffList)
# Since thatch limiting class factor, only take use first 1400 metal images
dataMetal = tiff_toArray(metal_tiffList[0:1400])
dataNoRoof = tiff_toArray(noroof_tiffList)

print ('Thatch = ', dataThatch.shape)
print ('Metal = ', dataMetal.shape)
print ('No Roof = ', dataNoRoof.shape)


# Remove duplicate images
dataThatch = np.unique(dataThatch, axis=0)
dataMetal = np.unique(dataMetal, axis=0)
dataNoRoof = np.unique(dataNoRoof, axis=0)

print ('Unique data sizes:')
print ('Thatch data shape = ', dataThatch.shape)
print ('Metal data shape = ', dataMetal.shape)
print ('NoRoof data shape = ', dataNoRoof.shape)

# Make sure thatch and metal samples are the same size and noroof samples are
# (total size - amount to be 'stretched')
lim_factor_size = dataThatch.shape[0]
dataThatch = dataThatch[0:lim_factor_size]
dataMetal = dataMetal[0:lim_factor_size]
dataNoRoof = dataNoRoof[0:(lim_factor_size*3)]

# Subset of data for brightness/saturation augmentation
dataNoRoof_forAug = dataNoRoof[0:(lim_factor_size)]

print ('Thatch data shape = ', dataThatch.shape)
print ('Metal data shape = ', dataMetal.shape)
print ('NoRoof total data shape = ', dataNoRoof.shape)

# Get Statistics and perform Histogram Stretch
def histogram_Stats(img):
  pix_min = np.min(img)
  pix_max = np.max(img)
  breadth = pix_max - pix_min
  return breadth                    # 0.156 - 0.014  for all data

def histogram_Stretch(img,imgAll):
  pix_min = np.min(img)
  pix_max = np.max(img)
  stretch_factor = 1.31    # Range of reflectance values from Zimbabwe
  stretch=((img - pix_min)/(pix_max - pix_min))*histogram_Stats(imgAll)*stretch_fac
tor
  return stretch

# Augmentation
def Augment(data_size, data, allAug):
  flipped_data = np.zeros([data_size,img_size,img_size,4], dtype=float)
  rotated_data = np.zeros([data_size,img_size,img_size,4], dtype=float)
```

```
  stretch_data = np.zeros([data_size,img_size,img_size,4], dtype=float)
  f_count = 0
  r_count = 0
  s_count = 0

  # Horizontally flip, rotate by 90 degrees and stretch data
  for i in data:
    flipped = tf.image.flip_left_right(i)
    rotated = tf.image.rot90(i)
    stretched = histogram_Stretch(i,dataThatch) #Thatch chosen for largest
value range

    flipped_data[f_count,:,:,:] = flipped
    f_count += 1
    rotated_data[r_count,:,:,:] = rotated
    r_count += 1
    stretch_data[s_count,:,:,:] = stretched
    s_count += 1

  # If allAug set to True, append all types of augmentation.
  # Otherwise just append the flipped and rotated
  if allAug == True:
    aug_data = np.concatenate((flipped_data, rotated_data, stretch_data), axis = 0)
  else:
    aug_data = stretch_data
  return (aug_data)

# Run Augment function on thatch and metal datasets.
orig_data_size = dataThatch.shape[0]
augdata_thatch = Augment(orig_data_size, dataThatch[...,:4], allAug=True)
augdata_metal = Augment(orig_data_size, dataMetal[...,:4], allAug=True)

# Visualize data augmentations
im = 300
flip = im
rot = 346 + im
stre = 346 + rot
band = 3             # Choose which color band to show (0,1,2,3 = r,g,b,NIR)

fig = plt.figure(figsize=(10,10))
fig.add_subplot(2,2,1)
plt.imshow(dataMetal[im][:,:,band])        # Original
plt.title('Original')
plt.colorbar()

fig.add_subplot(2,2,2)
plt.imshow(augdata_metal[flip][:,:,band])    # Flipped
plt.title('Flipped')
plt.colorbar()

fig.add_subplot(2,2,3)
plt.imshow(augdata_metal[rot][:,:,band])    # Rotated
plt.title('Rotated')
plt.colorbar()

# Make color range the same for all so that histogram stretch can be visualized
fig.add_subplot(2,2,4)
plt.imshow(augdata_metal[stre][:,:,band], vmin=.03, vmax=0.05)    # Stretched
plt.title('Stretched')
plt.colorbar()
plt.show()

#Save figure as png
```

```
fig.savefig('/content/drive/My Drive/UoE_GIS/FIGURES/Augments.png')

# Combine all data (ORDER CONSISTENCY MATTERS)
DATA = np.concatenate((dataNoRoof, augdata_noroof, dataMetal, augdata_metal, dataTh
atch, augdata_thatch), axis=0)
print ('All DATA shape: ', DATA.shape)


# Function to create label vectors from label files that include class info
# and bounding box info. Mostly for object detection (unnecessary for
classification)
def createLabels(class_name, labelList):
  labels = np.zeros([len(labelList),7], dtype=float)
  xmin = 0
  ymin = 0
  xmax = 0
  ymax = 0
  ind = 0
  count = 0

  # Loop through text files and get bounding box info
  for label in labelList:
    labelfile = open(label, "r")
    lines = labelfile.readlines()

    for l in lines:
      if '<xmin>' in l:
        l = l.split('>')
        ind = l[1].find('<')
        xmin = float(l[1][0:ind])
      if '<ymin>' in l:
        l = l.split('>')
        ind = l[1].find('<')
        ymin = float(l[1][0:ind])
      if '<xmax>' in l:
        l = l.split('>')
        ind = l[1].find('<')
        xmax = float(l[1][0:ind])
      if '<ymax>' in l:
        l = l.split('>')
        ind = l[1].find('<')
        ymax = float(l[1][0:ind])

    # Calculate midpoint, width, and height of each bounding box
    width = (xmax-xmin)
    height = (ymax-ymin)
    xmid = (width/2) + xmin
    ymid = (height/2) + ymin

    # Append label and bounding box info to LABEL vector, and put into list of labe
ls
    labels[count][0] = class_name
    labels[count][1] = xmid
    labels[count][2] = ymid
    labels[count][3] = width
    labels[count][4] = height
    count += 1
  return (labels)

# Run createLabels function for training data (takes about 30min)
labelsThatch = createLabels(2.0, thatch_labelList)
labelsMetal = createLabels(1.0, metal_labelList[0:1400])
print ('Labels created')
```

```
# Create background
labels (0's) data with shape 7 for compatibility with other labels
labelsNoRoof = np.zeros([(lim_factor_size * 4),7], dtype=float)
for l in labelsNoRoof:
  l[0] = 0.0
print (labelsNoRoof.shape)

# Remove duplicate labels and make same size
labels_thatch = np.unique(labelsThatch, axis=0)
labels_metal = np.unique(labelsMetal, axis=0)
labels_metal = labels_metal[0:lim_factor_size]
labels_thatch = labels_thatch[0:lim_factor_size]
print (labels_thatch.shape)
print (labels_metal.shape)

# Augment Labels
def augmentLabels(labelslist):

  xflip = []
  new_x = []
  new_y = []
  new_width = []
  new_height = []

  # Deep copies used so original labels preserved
  flip_labels = copy.deepcopy(labelslist)
  rotate_labels = copy.deepcopy(labelslist)
  stretch_labels = copy.deepcopy(labelslist)

  # Horizontal Flip
  for l in flip_labels:
    xflip.append(img_size - l[1])
    for a in range(len(xflip)):
      flip_labels[a][1] = xflip[a]

  # Counterclockwise Rotation (90 degrees)
  for l in rotate_labels:
    new_x.append(img_size - l[2])
    new_y.append(l[1])
    new_width.append(l[4])
    new_height.append(l[3])
    for a in range(len(new_x)):
      rotate_labels[a][1] = new_x[a]
      rotate_labels[a][2] = new_y[a]
      rotate_labels[a][3] = new_width[a]
      rotate_labels[a][4] = new_height[a]

  aug_labels = np.concatenate((flip_labels,rotate_labels,stretch_labels), axis=0)
  return(aug_labels)

# Run augment function on thatch and metal datasets
auglabels_thatch = augmentLabels(labels_thatch)
auglabels_metal = augmentLabels(labels_metal)
print (auglabels_thatch.shape)
print (auglabels_metal.shape)

# Visualize label augmentations
im = 200
flip = im
rot = 346 + im
stre = 346 + rot
```

```
L = labels_metal[im]                        # Original
xc = L[1]
yc = L[2]
w = L[3]
h = L[4]
xmin = (xc - (w/2))
ymin = (yc - (h/2))
plt.imshow(dataMetal[im][:,:,0])
tbox = patches.Rectangle((xmin, ymin),
        w,
        h,
        linewidth=2,
        edgecolor = 'red',
        fill=False)
plt.gca().add_patch(tbox)
plt.title('Original')
plt.show()
print (L)


L = auglabels_metal[flip]                     # Flipped
xc = L[1]
yc = L[2]
w = L[3]
h = L[4]
xmin = (xc - (w/2))
ymin = (yc - (h/2))
plt.imshow(augdata_metal[flip][:,:,0])
tbox = patches.Rectangle((xmin, ymin),
        w,
        h,
        linewidth=2,
        edgecolor = 'red',
        fill=False)
plt.gca().add_patch(tbox)
plt.title('Flipped')
plt.colorbar()
plt.show()
print (L)

L = auglabels_metal[rot]                     # Rotated
xc = L[1]
yc = L[2]
w = L[3]
h = L[4]
xmin = (xc - (w/2))
ymin = (yc - (h/2))
plt.imshow(augdata_metal[rot][:,:,0])
tbox = patches.Rectangle((xmin, ymin),
        w,
        h,
        linewidth=2,
        edgecolor = 'red',
        fill=False)
plt.gca().add_patch(tbox)
plt.title('Rotated')
plt.colorbar()
plt.show()
print (L)

# Combine all labels and make sure size matches data
LABELS = np.concatenate((labelsNoRoof, labels_metal, auglabels_metal, labels_thatch
, auglabels_thatch), axis=0)
```

```
print ('Final LABEL shape: ', LABELS.shape)
print ('Final DATA shape: ', DATA.shape)
# Save DATA and LABELS arrays to h5py file.
with h5py.File('/content/drive/My Drive/To_Gary/Arrays_10.h5', 'w') as hf:
  hf.create_dataset('DATA', data=DATA)
  hf.create_dataset('LABELS', data=LABELS)
```

II: Training Classifier Script

```
# TRAIN_CLASSIFIRE.ipynb
# Imports
import h5py
import numpy as np
from osgeo import gdal
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import scipy
from scipy.ndimage import rotate
sns.set(color_codes=True)
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
import tensorflow as tf
from tensorflow import keras
%matplotlib inline
# Keras Imports
import keras
import keras.utils
from keras.preprocessing.image import load_img
from keras.models import load_model
from keras.layers import Input, Flatten, Activation, Dense, Dropout
from keras.layers import Conv2D, MaxPooling2D, GlobalMaxPooling2D
from keras.models import Model, Sequential
from keras.callbacks import EarlyStopping, ReduceLROnPlateau
from keras import optimizers
from keras import backend as K


# Read training data and labels from file (Arrays_1,2,3,etc.)
data_filepath = '/content/drive/My Drive/To_Gary/Arrays_10.h5'
with h5py.File(data_filepath, 'r') as hf:
  DATA = hf['DATA'][:]
  LABELS = hf['LABELS'][:]


# Function to randomly shuffle DATA and LABELS so their indices still correspond
def shuffle_in_unison(a, b):
    rng_state = np.random.get_state()
    np.random.shuffle(a)
    np.random.set_state(rng_state)
    np.random.shuffle(b)

# Shuffle
shuffle_in_unison(DATA,LABELS)
print ('Data shuffled')
print ('Data shape: ', DATA.shape)
print ('Labels shape: ', LABELS.shape)

# Split data into training and testing
test_size = 0.10 * DATA.shape[0]    # 10% of total data
test_size = int(test_size)
print ('Number of testing images = ' + str(test_size))
```

```
train_size = DATA.shape[0] - test_size   # total data - test data
print ('Number of training images = ' + str(train_size))

test_data = DATA[:test_size,:,:,:]
train_data = DATA[test_size:,:,:,:]
print ('Shape of test data: ', test_data.shape)
print ('Shape of train data: ', train_data.shape)

# Use only class info (not bounding box)
test_labels = LABELS[:test_size,0]
train_labels = LABELS[test_size:,0]
print('Shape of test labels: ', test_labels.shape)
print('Shape of train labels: ', train_labels.shape)

# Count classes in gound truth data to make sure relatively even distribution
thatch_count_A = 0
metal_count_A = 0
noroof_count_A = 0
for a in test_labels:
  if a == 1:
    metal_count_A += 1
  elif a == 2:
    thatch_count_A += 1
  else:
    noroof_count_A += 1
print ('Actual Metal buildings = ', metal_count_A)
print ('Actual Thatch buildings = ', thatch_count_A)
print ('Actual Noroof samples = ', noroof_count_A)

# Load pre-trained model from file (if want to continue training)
'''model_filepath = '/content/drive/My Drive/To_Gary/model_5.h5'
model_continued = load_model(model_filepath)
print (model_continued.summary())'''

# Build simple classifier model from scratch (3 Convolutions)
img_size = 30

model_3Convs = Sequential()
model_3Convs.add(Conv2D(32, kernel_size=(3, 3), strides=(1,1),
                 activation='relu',
                 input_shape=(img_size,img_size,4)))
model_3Convs.add(Conv2D(64, (3, 3), activation='relu'))
model_3Convs.add(Conv2D(128, (3, 3), activation='relu'))
model_3Convs.add(GlobalMaxPooling2D())
model_3Convs.add(Dropout(0.25))
model_3Convs.add(Dense(256, activation='relu'))
model_3Convs.add(Dropout(0.5))
model_3Convs.add(Dense(3, activation='softmax'))
print(model_3Convs.summary())

# Compile model and set EarlyStopping (once loss stops decreasing, stop training)
epochs = 100
early_stop = EarlyStopping(monitor='loss', patience=30, restore_best_weights=False)
model_3Convs.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Train/fit model (If you run this more than once without recompiling model,
# it will pick up training where it left off)
model_3Convs.fit(train_data, train_labels, epochs=epochs, shuffle=True, validation_
data=(test_data, test_labels), callbacks=[early_stop])
```

```
# Evaluate model
test_loss, test_acc = model_3Convs.evaluate(test_data, test_labels, verbose=1)
print('Test accuracy:', test_acc*100, '%')

# Show Training History
history = model_3Convs.fit(train_data, train_labels, validation_data=(test_data, te
st_labels), epochs=epochs, verbose=0)
# list all data in history
print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

# Create predictions for test data
predictions = model_3Convs.predict(test_data)

# Save max predictions (classifications) to list and also create corresponding
string # lists for true and predicted labels for comparison.
predict_labels = []
predict_strings = []
true_strings = []

for p in predictions:
  l = np.argmax(p)
  predict_labels.append(l)
  if l == 0:
    predict_strings.append('No roofs')
  elif l == 1:
    predict_strings.append('Metal')
  elif l == 2:
    predict_strings.append('Thatch')

for t in test_labels:
  if t == 0:
    true_strings.append('No roofs')
  elif t == 1:
    true_strings.append('Metal')
  elif t == 2:
    true_strings.append('Thatch')

print ('No Roof label = 0')
print ('Metal label = 1')
print ('Thatch label = 2')
print (predict_labels)
print (predict_strings)
print (true_strings)

# Visualization (good cmaps = Spectra, twilight, Blue-r, coolwarm, hsv, ocean, summ
er)
# https://matplotlib.org/3.1.0/tutorials/colors/colormaps.html
```

```
print ('Size of test data = ', test_data.shape)

# Choose any two images out of test data size for comparison
img1 = 300
img2 = 250

plt.imshow(test_data[img1][:,:,0], cmap='ocean')        # img1
plt.title('Actual label = ' + str(true_strings[img1]))
plt.text(40,0, 'Predicted = ' + str(predict_strings[img1]))
plt.colorbar()
plt.grid(b=None)
plt.show()

plt.imshow(test_data[img2][:,:,0], cmap='ocean')        # img2
plt.title('Actual label = ' + str(true_strings[img2]))
plt.text(40,0, 'Predicted = ' + str(predict_strings[img2]))
plt.colorbar()
plt.grid(b=None)
plt.show()

# Visualization of wrongly predicted buildings
index = 0
wrong = []
for t in range(len(test_labels)):
  if test_labels[t] != predict_labels[t]:
    wrong.append(index)
  index += 1

for w in wrong:
  plt.imshow(test_data[w][:,:,3], cmap='ocean')
  plt.title('Img: ' + str(w))
  plt.text(40,0, 'Actual = ' + str(true_strings[w]))
  plt.text(40,3, 'Predicted = ' + str(predict_strings[w]))
  plt.colorbar()
  plt.grid(b=None)
  plt.show()

# Count buildings in predictions
thatch_count_P = 0
metal_count_P = 0
noroof_count_P = 0
for p in predict_labels:
  if p == 1:
    metal_count_P += 1
  elif p == 2:
    thatch_count_P += 1
  else:
    noroof_count_P += 1

# Count buildings in gound truth data
thatch_count_A = 0
metal_count_A = 0
noroof_count_A = 0
for a in test_labels:
  if a == 1:
    metal_count_A += 1
  elif a == 2:
    thatch_count_A += 1
  else:
    noroof_count_A += 1

print ('Predicted Metal buildings = ', metal_count_P)
print ('Predicted Thatch buildings = ', thatch_count_P)
```

```
print ('Predicted Noroof samples = ', noroof_count_P)

print ('Actual Metal buildings = ', metal_count_A)
print ('Actual Thatch buildings = ', thatch_count_A)
print ('Actual Noroof samples = ', noroof_count_A)

# Calculate model accuracies for each class
print ('Metal accuracy = ', (metal_count_P/metal_count_A)*100, '%')
print ('Thatch accuracy = ', (thatch_count_P/thatch_count_A)*100, '%')
print ('Noroof accuracy = ', (noroof_count_P/noroof_count_A)*100, '%')

# Calculate number of wrongly-classified images
wrong_count = 0
for i in range(len(test_labels)):
  if test_labels[i] != predict_labels[i]:
    wrong_count += 1
print ('Number of total wrongly predicted images:', wrong_count)

# Compute confusion matrix
actual = test_labels
predicted = predict_labels
results = confusion_matrix(actual, predicted)

print ('Confusion Matrix :')
print (results)
print ('Accuracy Score : ',accuracy_score(actual, predicted))
print ('Report : ')
print (classification_report(actual, predicted))

#Once satisfactory accuracy is met, save model architecture & weights or just
weights
model_3Convs.save('/content/drive/My Drive/To_Gary/model_12_3Conv.h5')
model_3Convs.save_weights('/content/drive/My Drive/To_Gary/model_weights_8.h5')
```

III: Object Detection Script

```
# OBJECT_DETECTOR.ipynb
# Imports
import copy
import time
import h5py
import numpy as np
from numpy.random import random
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import scipy
import seaborn as sns
from scipy.ndimage import rotate
sns.set(color_codes=True)
import tensorflow as tf
from tensorflow import keras
%matplotlib inline
# Keras Imports (must have tensorflow as backend)
import keras
import tensorflow.keras.utils
```

```
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras import layers
from tensorflow.keras.layers import Input, Flatten, Activation, Dense, Dropout
from tensorflow.keras.layers import Conv2D, MaxPooling2D, GlobalMaxPooling2D
from tensorflow.keras.layers import LeakyReLU
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.models import Model
from tensorflow.keras.models import Sequential
from tensorflow.keras.models import load_model
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import ReduceLROnPlateau
from tensorflow.keras import optimizers
from tensorflow.keras import backend as K


# Load pre-trained model and the associated data to keep training (if already
started)
# If this is first time, skip this part
#model_filepath = '/content/drive/My Drive/To_Gary/model_ObjDet.h5'
#data_filepath = '/content/drive/My Drive/To_Gary/Arrays_8.h5'
#weights_filepath = '/content/drive/My Drive/To_Gary/model_weights_8.h5'
#model = load_model(model_filepath)

with h5py.File(data_filepath, 'r') as hf:
  DATA = hf['DATA'][:]
  LABELS = hf['LABELS'][:]

print (model.summary())
print ('Data shape: ', DATA.shape, DATA.dtype)
print ('Labels shape:', LABELS.shape, LABELS.dtype)

# Print current accuracy of model
# model_loss, model_acc = model.evaluate(DATA, LABELS, verbose=1)
# print('Train accuracy:', model_acc*100, '%')

# Change all NaN values to 0 so that model can still perform computations and one-h
ot encode class info
# Change last 2 values to 'one-hot encoded labels'
for L in LABELS:
  if L[0] == 0:
    L[:] = 0.0
  elif L[0] == 1:
    L[5] = 1.0
  else:
    L[6] = 1.0

# Change first value to 0 or 1 to identify whether a roof is present or not
LABELS[:,0] = np.where(LABELS[:,0] == 2.0, np.ones_like(LABELS[:,0]), LABELS[:,0])

# Normalize box info (from between 0 and 1)
img_size = DATA.shape[1]
for L in LABELS:
  if L[0] == 1:
    L[1] = L[1]/img_size
    L[2] = L[2]/img_size
    L[3] = L[3]/img_size
    L[4] = L[4]/img_size

# Function to randomly shuffle DATA and LABELS so their indices still correspond
def shuffle_in_unison(a, b):
    rng_state = np.random.get_state()
    np.random.shuffle(a)
    np.random.set_state(rng_state)
```

```
    np.random.shuffle(b)
# Shuffle
shuffle_in_unison(DATA,LABELS)
print ('Data shuffled')
print ('Data shape: ', DATA.shape)
print ('Labels shape: ', LABELS.shape)

# Split data into training and testing
test_size = 0.10 * DATA.shape[0]    # 10% of total data
test_size = int(test_size)
train_size = DATA.shape[0] - test_size   # total data - test data
test_data = DATA[:test_size,:,:,:]
train_data = DATA[test_size:,:,:,:]
print('Shape of test data: ', test_data.shape)
print('Shape of train data: ', train_data.shape)
# Split labels into training and testing
test_labels = LABELS[:test_size,:]
train_labels = LABELS[test_size:,:]
print('Shape of test labels: ', test_labels.shape)
print('Shape of train labels: ', train_labels.shape)


# Build object detection model with leaky relu implementation
img_size = 30

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), kernel_initializer=tf.keras.initializers.o
nes,
                 input_shape=(img_size,img_size,4)))
model.add(LeakyReLU(alpha=0.01))
model.add(Conv2D(64, (3, 3)))
model.add(LeakyReLU(alpha=0.01))
model.add(GlobalMaxPooling2D())
model.add(Dropout(0.25))
model.add(Dense(128))
model.add(LeakyReLU(alpha=0.01))
model.add(Dropout(0.5))
model.add(Dense(7, activation='softmax'))

print(model.summary())

# Custom loss function based on YOLO, takes bounding box info as well as class info
@tf.function
def custom_loss(y_true,y_pred):

  # Create object/noobject mask
  obj = y_true[:,0]
  noobj = tf.where((obj == 0.0), 1.0, 0.0)  # Switch values

  # Weights for different label parameters
  penalty_box = tf.constant(5.0)
  penalty_class = tf.constant(2.0)
  penalty_obj = tf.constant(5.0)
  penalty_noobj = tf.constant(0.5)

  # Calculate corners of bounding boxes
  xminT = y_true[:,1] - (y_true[:,3]/2)
  xmaxT = y_true[:,1] + (y_true[:,3]/2)
  xminP = y_pred[:,1] - (y_pred[:,3]/2)
  xmaxP = y_pred[:,1] + (y_pred[:,3]/2)

  yminT = y_true[:,2] - (y_true[:,4]/2)
  ymaxT = y_true[:,2] + (y_true[:,4]/2)
```

```
  yminP = y_pred[:,2] - (y_pred[:,4]/2)
  ymaxP = y_pred[:,2] + (y_pred[:,4]/2)

  # Calculate corners of Intersection of boxes
  interXmin = tf.maximum(xminT, xminP)
  interXmax = tf.minimum(xmaxT, xmaxP)
  interYmin = tf.maximum(yminT, yminP)
  interYmax = tf.minimum(ymaxT, ymaxP)

  # Calculate width and height of inner/Intersection box
  interW = interXmax - interXmin
  interH = interYmax - interYmin

  # Calculate Intersection over Union (IOU) between true and predicted bounding box
es
  interArea = interW * interH
  totalArea = (y_true[:,3]*y_true[:,4]) + (y_pred[:,3]*y_pred[:,4]) - interArea
  IOU = tf.truediv(interArea,totalArea)
  IOU = tf.where(tf.math.is_nan(IOU),tf.constant(0.0), IOU)
#Convert nan values to 0.0

  # Calculate loss values for position and size of object, confidence of object
  # localization based on IOU, and confidence that object is classified correctly
  coord_ls = obj * penalty_box * tf.sqrt(tf.divide(tf.reduce_sum(tf.square(tf.subtr
act    (y_true[:,1:2],y_pred[:,1:2]))),tf.cast(tf.size(y_true[:,0]),tf.float32)))
  size_ls = obj * penalty_box * tf.sqrt(tf.divide(tf.reduce_sum(tf.square(tf.subtra
ct(
tf.sqrt(y_true[:,3:4]),tf.sqrt(y_pred[:,3:4])))),tf.cast(tf.size(y_true[:,0]),t
f.float32)))
  conf_ls = (obj * tf.sqrt(tf.divide(tf.reduce_sum(tf.square(tf.subtract(1.0,IOU)))
,tf
.cast(tf.size(obj),tf.float32)))) + (noobj * penalty_noobj * tf.sqrt(tf.divide(
tf.reduce_sum(tf.square(tf.subtract(1.0,IOU))),tf.cast(tf.size(noobj),tf.float3
2))))
  class_ls = penalty_class * tf.keras.losses.binary_crossentropy(y_true[:,5:6], y_p
red    [:,5:6], from_logits=True)

  loss = coord_ls + size_ls + conf_ls + class_ls
  return loss

# Create custom forward and backward passes
@tf.function
def forward_pass(model, inputs, target):
  with tf.GradientTape(persistent=True) as tape:
    target_pred = model(inputs)
    loss = custom_loss(target,target_pred)
  grads = tape.gradient(loss, model.trainable_variables)
  return loss, grads

@tf.function
def backward_pass(model, grads, optimizer):
  optimizer.apply_gradients(zip(grads,model.trainable_variables))

# Convert data to tensors for input into model
train_data = tf.convert_to_tensor(train_data)
train_labels = tf.convert_to_tensor(train_labels)
test_data = tf.convert_to_tensor(test_data)
test_labels = tf.convert_to_tensor(test_labels)

# Put tensors into tf.Dataset for shuffling purposes
train_dataset = tf.data.Dataset.from_tensor_slices((train_data,train_labels))
test_dataset = tf.data.Dataset.from_tensor_slices((test_data,test_labels))
```

```
# Training Loop
epochs = 250
#optimizer = tf.keras.optimizers.Adam(lr=.001)
optimizer = tf.keras.optimizers.Adam(lr=0.5e-4, beta_1=0.9, beta_2=0.999, epsilon=1
e-05, decay=0.0001)

for epoch in range(epochs):
  print("\nStart of epoch %d" % (epoch + 1))
  epochStart = time.time()
  train_dataset = train_dataset.shuffle(buffer_size=3737).batch(101)         # Shuff
le and split into batches
  for step, (x_batch_train, y_batch_train) in enumerate(train_dataset):

    loss, grads = forward_pass(model, train_data, train_labels)
    backward_pass(model, grads, optimizer)

 #Evaluate at the end of each epoch
  train_pred = model(train_data)
  test_pred = model(test_data)
  train_loss = custom_loss(train_labels, train_pred)
  test_loss = custom_loss(test_labels, test_pred)
  train_loss_avg = tf.reduce_mean(train_loss)
  test_loss_avg = tf.reduce_mean(test_loss)
  print('Train Validation Loss: %.3f' % train_loss_avg)
  print('Test Validation Loss: %.3f' % test_loss_avg)
  epochEnd = time.time()
  elapsed = (epochEnd - epochStart) / 60.0
  print('took %.4f minutes' % elapsed)


# Evaluate model on train and test data
train_loss, train_acc = model.evaluate(train_data, train_labels, verbose=1)
print('Train accuracy:', train_acc*100, '%')
test_loss, test_acc = model.evaluate(test_data, test_labels, verbose=1)
print('Test accuracy:', test_acc*100, '%')

# Test labels outputs
predictions = model.predict(train_data)

#OPTIONAL: Once satisfactory accuracy is met, save model architecture and weights
model.save('/content/drive/My Drive/To_Gary/model_ObjDet.h5')
```

IV: Testing Classifier Script

```
# CLASIFIER_TESTING.ipynb
# Imports
import os
import time
import copy
import h5py
import numpy as np
from osgeo import gdal

import matplotlib
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle
import tensorflow as tf
from tensorflow import keras
# Keras Imports
import keras
import tensorflow.keras.utils
from tensorflow.keras.preprocessing.image import load_img
```

```
from tensorflow.keras import layers
from tensorflow.keras.layers import Input, Flatten, Activation, Dense, Dropout
from tensorflow.keras.layers import Conv2D, MaxPooling2D, GlobalMaxPooling2D
from tensorflow.keras.layers import LeakyReLU
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.models import Model
from tensorflow.keras.models import Sequential
from tensorflow.keras.models import load_model
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import ReduceLROnPlateau
from tensorflow.keras import optimizers
from tensorflow.keras import backend as K

# Load pre-trained model and the data it trained on to see accuracy
model_filepath = '/content/drive/My Drive/To_Gary/model_12_3Conv.h5'
data_filepath = '/content/drive/My Drive/To_Gary/Arrays_8.h5'
weights_filepath = '/content/drive/My Drive/To_Gary/model_weights_8.h5'
model = load_model(model_filepath)

with h5py.File(data_filepath, 'r') as hf:
  DATA = hf['DATA'][:]
  LABELS = hf['LABELS'][:]

print (model.summary())
print ('Data shape: ', DATA.shape, DATA.dtype)
print ('Labels shape:', LABELS.shape, LABELS.dtype)

# Print accuracy of model on the data it trained on
model_loss, model_acc = model.evaluate(DATA, LABELS[:,0], verbose=1)
print('Train accuracy:', model_acc*100, '%')

# Path to tiff image to run through classifier
root = '/content/drive/My Drive/UoE_GIS/Zimbabwe/Anja_Sub1.tif'
img_size = 30
# Function to read a tiff into one 3D array instead of in 30x30 chunks (no batches)
def Entire_Bigtiff(tiff):
  ds=gdal.Open(tiff)
  # read data from geotiff object
  numX=ds.RasterXSize                    # number of pixels in x direction
  numY=ds.RasterYSize                    # number of pixels in y direction
  print('Total pixels = ', numX,numY)
  # geolocation tiepoint
  transform_ds = ds.GetGeoTransform() # extract geolocation information
  xOrigin=transform_ds[0]             # coordinate of x corner
  yOrigin=transform_ds[3]             # coordinate of y corner
  pixelWidth=transform_ds[1]          # resolution in x direction
  pixelHeight=transform_ds[5]         # resolution in y direction

  # Geospatial information for later writing outputs to tiff
  geotransform = (xOrigin, pixelWidth, 0, yOrigin, 0, pixelHeight)
  # Create array of zeros with proper shape
  data = np.zeros([numY,numX,4], dtype=float)

  # read data. Returns as a 2D numpy array
  NIR=ds.GetRasterBand(4).ReadAsArray()
  blue=ds.GetRasterBand(3).ReadAsArray()
  green=ds.GetRasterBand(2).ReadAsArray()
  red=ds.GetRasterBand(1).ReadAsArray()

  # Insert band values into proper place in array
  data[:,:,3] = NIR
  data[:,:,2] = blue
  data[:,:,1] = green
```

```
  data[:,:,0] = red
  return (data, geotransform, numX, numY)

# Call on above function
M, M_geotransform, numX, numY = Entire_Bigtiff(root)
print(M_geotransform)

# Function to split big tiff into batches and convert to 4D numpy array (batches, i
mg_# size, img_size, bands)
def Bigtiff_toArray(tiff):
  step_size = img_size
  count = 0
  ds=gdal.Open(tiff)
  # read data from geotiff object
  numX=ds.RasterXSize             # number of pixels in x direction
  numY=ds.RasterYSize             # number of pixels in y direction
  print('Total pixels = ', numX,numY)
  # Batch size of rows/columns
  batches_in_row = int(numX/step_size)
  batches_in_col = int(numY/step_size)
  batches = batches_in_row * batches_in_col
  print('Number of subsets in x and y direction = ', batches_in_row, batches_in_col
)
  print('Number of total subsets/images produced from tiff = ', batches)

  # geolocation tiepoint
  transform_ds = ds.GetGeoTransform()# extract geolocation information
  xOrigin=transform_ds[0]       # coordinate of x corner
  yOrigin=transform_ds[3]       # coordinate of y corner
  pixelWidth=transform_ds[1]    # resolution in x direction
  pixelHeight=transform_ds[5]   # resolution in y direction

  # Create array of zeros with proper shape
  data = np.zeros([batches,step_size,step_size,4], dtype=float)

  for row in range(batches_in_row):         # -1 for shift right
    for col in range(batches_in_col):       # -1 for shift down

      # Read data into array
      # Add 15 to xoff or yoff for shift right/down
      NIR=ds.GetRasterBand(4).ReadAsArray(xoff=(row*step_size), yoff=(col*step_size
),                win_xsize=step_size, win_ysize=step_size)
      blue=ds.GetRasterBand(3).ReadAsArray(xoff=(row*step_size), yoff=(col*step_siz
e),          win_xsize=step_size, win_ysize=step_size)
      green=ds.GetRasterBand(2).ReadAsArray(xoff=(row*step_size), yoff=(col*step_si
ze)                , win_xsize=step_size, win_ysize=step_size)
      red=ds.GetRasterBand(1).ReadAsArray(xoff=(row*step_size), yoff=(col*step_size
),                win_xsize=step_size, win_ysize=step_size)

      # Insert band values into proper place in array
      data[count,:,:,3] = NIR
      data[count,:,:,2] = blue
      data[count,:,:,1] = green
      data[count,:,:,0] = red
      count += 1
  return (data, batches_in_row, batches_in_col)

# Call on function to convert tiff to batched 4D numpy array
data, row, col = Bigtiff_toArray(root)
print('Shape of data: ', data.shape)

# From saved model, create predictions list on big dataset and convert to
labels(max prediction)
```

```
predictions = model.predict(data)
predict_labels = []
for p in predictions:
  predict_labels.append(np.argmax(p))

# From predictions, count metal and thatch buildings
thatch_count = 0
metal_count = 0
noroof_count = 0
for p in predict_labels:
  if p == 1:
    metal_count += 1
  elif p == 2:
    thatch_count += 1
  else:
    noroof_count += 1
print('Predicted thatch buildings = ', thatch_count)
print('Predicted metal buildings = ', metal_count)

# Get indices of where buildings are
m_ind = []
t_ind = []
for p in range(len(predict_labels)):
  if predict_labels[p]==1:
    m_ind.append(p)
  elif predict_labels[p]==2:
    t_ind.append(p)
print(m_ind)
print(t_ind)
print(len(m_ind),len(t_ind))

# Create LABEL MASK and populate with label data
count = 0
label_Mask = np.zeros([data.shape[0],img_size,img_size], dtype=float)
for l in predict_labels:
  label_Mask[count,:,:].fill(l)
  count += 1
# Merge labels in proper shape for output tiff
subset_x = row
subset_y = col
window = img_size
start = 0
end = img_size
count = 0
Merge = np.zeros([numY,numX], dtype=float)
for i in range(subset_x):
  for j in range(subset_y):
    x0 = start + (j*window)
    y0 = start + (i*window)
    x1 = start + ((j+1)*window)
    y1 = start + ((i+1)*window)

    Merge[x0:x1,y0:y1] = label_Mask[count,:,:]
    count += 1

# OPTIONAL: Create PROBABILITY MASK and populate with prediction data
'''count = 0
probability_Mask = np.zeros([data.shape[0],img_size,img_size,3], dtype=float)
for l in predictions:
  probability_Mask[count,:,:,0].fill(l[0])
  probability_Mask[count,:,:,1].fill(l[1])
  probability_Mask[count,:,:,2].fill(l[2])
  count += 1
```

```
# Merge probabilities into proper shape for output tiff
subset_x = 45
subset_y = 45
window = img_size
start = 0
end = img_size
count = 0
Merge = np.zeros([numY,numX,3], dtype=float)
for i in range(subset_x):
  for j in range(subset_y):
    x0 = start + (j*window)
    y0 = start + (i*window)
    x1 = start + ((j+1)*window)
    y1 = start + ((i+1)*window)

    Merge[x0:x1,y0:y1,:] = probability_Mask[count,:,:,:]
    count += 1'''

# Visualize label_Mask output next to actual image
fig=plt.figure(figsize=(10,10))
plt.imshow(Merge)
plt.colorbar()
plt.show()

fig=plt.figure(figsize=(10,10))
plt.imshow(M[:,:,3])
plt.show()

# Write Label_Mask to tiff
filename = '/content/drive/My Drive/UoE_GIS/Predictions/Zimb_labelMAsk.tif'
# load data in to geotiff object
ds = gdal.GetDriverByName('GTiff').Create(filename, numX, numY, 1, gdal.GDT_Float32
)
ds.SetGeoTransform(M_geotransform)                  # specify coords
ds.GetRasterBand(1).WriteArray(Merge[:,:])          # write image to the raster
ds.FlushCache()                                     # write to disk
ds = None                                           # reset
print("Image written to", filename)

# OPTIONAL: Write Probability_Mask to tiff
'''filename = '/content/drive/My Drive/UoE_GIS/Prob_Masks/Mbola_07_Masks/M7_0_model
12_All.tif'
ds = gdal.GetDriverByName('GTiff').Create(filename, numX, numY, 3, gdal.GDT_Float32
)
ds.SetGeoTransform(M_geotransform)                  # specify coords
ds.GetRasterBand(1).WriteArray(Merge[:,:,0])        # write image to the raster
ds.GetRasterBand(2).WriteArray(Merge[:,:,1])
ds.GetRasterBand(3).WriteArray(Merge[:,:,2])
ds.FlushCache()                                     # write to disk
ds = None                                           # reset
print("Image written to",filename)'''

# Save predictions and predict_labels to file (Currently using Model_12_3Convs)
with h5py.File('/content/drive/My Drive/UoE_GIS/Predictions/Zimb_predictions.h5', '
w') as hf:
  hf.create_dataset('predictions', data=predictions)
  hf.create_dataset('predict_labels', data=predict_labels)
```

V: Segmentation Script

```
# BATCH_SEGMENTATION_LOOP.py
```

```
# Imports
import arcpy
from arcpy import env
from arcpy.sa import *
# Turn on Overwrite (overwrite intermediate files to save space)
arcpy.env.overwriteOutput = True
# Set environment to proper directory
env.workspace = "D:/Anja_Dissertation/BuildingRoofDetection/Tanzania/Segmentation/I
nputs/Villages_07"

# Create list of tiffs
tiffList = arcpy.ListRasters("*.tif", "TIF")

# Convert ending .TIF to lowercase
tiffsList = []
for tiff in tiffList:
  lwc_tiff = tiff.replace("TIF", "tif")
  tiffsList.append(lwc_tiff)

# Loop over tiffs and perform Make Raster Layer (to extract 3 of 4 bands) > Train I
SO Cluster Classifier > Classify Raster
count = 0
for tiff in tiffsList:
  print(tiff)
  Input_raster = tiff
  raster_3Bands = "D:/Anja_Dissertation/BuildingRoofDetection/Tanzania/Segmentation
/In    puts/raster_3Bands/only3_07_" + str(count) + ".tif"
  Output_signature_file = "D:/Anja_Dissertation/BuildingRoofDetection/Tanzania/Segm
ent    ation/ISO/iso_clustered.ecd"
  Output_raster = "D:/Anja_Dissertation/BuildingRoofDetection/Tanzania/Segmentation
/ra    sters_classified/classes_07_" + str(count) + ".tif"

  arcpy.MakeRasterLayer_management(Input_raster, raster_3Bands, "#", "#", "1;3;4")
  TrainIsoClusterClassifier(raster_3Bands, "20", Output_signature_file ,"#", "20",
"0",   "1", "COLOR; MEAN; RECTANGULARITY; COMPACTNESS")
  classifiedraster = ClassifyRaster(raster_3Bands, Output_signature_file, "#")
  classifiedraster.save(Output_raster)
  print(count)
  count += 1
```

VI: Post-Processing Script

```
# POSTPROCESS_VALIDATION.py
'''This script is meant to be run on the individual outputs of the raster calculato
r
function which multiplies the reclassified Segmentation result with the reclassifie
d
CNN Label Mask result(both of which have been reclassifed to 0 = NoRoof,
1 = Metal or Thatch roof)'''

# Imports
import arcpy
from arcpy import env
from arcpy.sa import *

# Turn on Overwrite
arcpy.env.overwriteOutput = True

# Set environment to proper directory
env.workspace = "D:/Anja_Dissertation/BuildingRoofDetection/Tanzania/Outputs/Calcul
ate    d_Predictions"
```

```
tiff = "D:/Anja_Dissertation/BuildingRoofDetection/Tanzania/Outputs/Calculated_Pred
ict    ions/Label_Outputs/RastCalc_12_1_metal.tif"
Input_raster = tiff

# Get filename for full path insertion
file_split = tiff.split('/')
name = file_split[-1]      # get string after last slash
shp_name = name[:-4] + ".shp"    # get rid of .tif end and add .shp
print (shp_name)

# Paths
to_Polygon = "D:/Anja_Dissertation/BuildingRoofDetection/Tanzania/Outputs/To_Polygo
n/"    + shp_name
Select = "D:/Anja_Dissertation/BuildingRoofDetection/Tanzania/Outputs/Select_by_Thr
esh    old/" + shp_name
Aggregate = "D:/Anja_Dissertation/BuildingRoofDetection/Tanzania/Outputs/Cluster_Pr
edi    ctions/" + shp_name
True_Positive = "D:/Anja_Dissertation/BuildingRoofDetection/Tanzania/Validation_Tes
ts/    Intersect_True_Positives/Metal_2012/TruePos" + shp_name
MisClassified = "D:/Anja_Dissertation/BuildingRoofDetection/Tanzania/Validation_Tes
ts/    Intersect_MissClassified/Metal_2012/MisClassified" + shp_name
True_sameClass = "D:/Anja_Dissertation/BuildingRoofDetection/Tanzania/Ground_truth/
Mbo    la_12/Vill1_truMetal.shp"
True_diffClass = "D:/Anja_Dissertation/BuildingRoofDetection/Tanzania/Ground_truth/
Mbo    la_12/Vill1_trueThatch.shp"

# Convert raster to polygon, select buildings, aggregate/smooth data
arcpy.RasterToPolygon_conversion(Input_raster, to_Polygon, "NO_SIMPLIFY", "VALUE")
arcpy.Select_analysis(to_Polygon, Select, '"value" == 1')
arcpy.AggregatePolygons_cartography(Select_by_Threshold, Aggregate, "4 Meters",
      "2 SquareMeters", "0 SquareMeters", "NON_ORTHOGONAL", "#", "#")
# Perform intersect to obtain True Positives and MisClassified buildings
arcpy.Intersect_analysis([Aggregate,True_sameClass],True_Positive, "ALL", "", "INPU
T")
arcpy.Intersect_analysis([Aggregate,True_diffClass],MisClassified, "ALL", "", "INPU
T")
print(count)
count += 1
```

VII: Result Tables

Table 2: Confusion Matrix Results for the 2007 Mbola dataset, with True Positives shown in darker green, Misclassified values shown in lighter green, False Positives shown in the left-most column, and False Negatives shown in the top row.

2007 Village 1	*No Roof*	*True Metal (247)*	*True Thatch (20)*	*Producer's Accuracy*
No Roof	N/A	129	19	N/A
Predicted Metal (144)	26	117	1	81.3 %
Predicted Thatch (2)	1	1	0	0 %
User's Accuracy	N/A	47.4 %	0 %	

2007 Village 2	*No Roof*	*True Metal (185)*	*True Thatch (27)*	*Producer's Accuracy*
No Roof	N/A	98	23	N/A
Predicted Metal (88)	-2	87	3	98.9 %
Predicted Thatch (4)	3	0	1	25.0 %
User's Accuracy	N/A	47.0 %	3.7 %	

2007 Village 3	*No Roof*	*True Metal (81)*	*True Thatch (10)*	*Producer's Accuracy*
No Roof	N/A	42	8	N/A
Predicted Metal (50)	11	38	1	76.0 %
Predicted Thatch (1)	0	1	0	0 %
User's Accuracy	N/A	47.0 %	0 %	

2007 Village 4	*No Roof*	*True Metal (110)*	*True Thatch (17)*	*Producer's Accuracy*
No Roof	N/A	17	13	N/A
Predicted Metal (136)	47	87	2	64.0 %
Predicted Thatch (21)	13	6	2	9.5 %
User's Accuracy	N/A	79.0 %	11.8 %	

Table 3: Confusion Matrix Results for the 2012 Mbola dataset, with True Positives shown in darker green, Misclassified values shown in lighter green, False Positives shown in the left-most column, and False Negatives shown in the top row.

2012 Village 1	*No Roof*	*True Metal (296)*	*True Thatch (25)*	*Producer's Accuracy*
No Roof	N/A	125	15	N/A
Predicted Metal (183)	17	165	1	90.2 %
Predicted Thatch (100)	85	6	9	9.0 %
User's Accuracy	N/A	55.7 %	36.0 %	

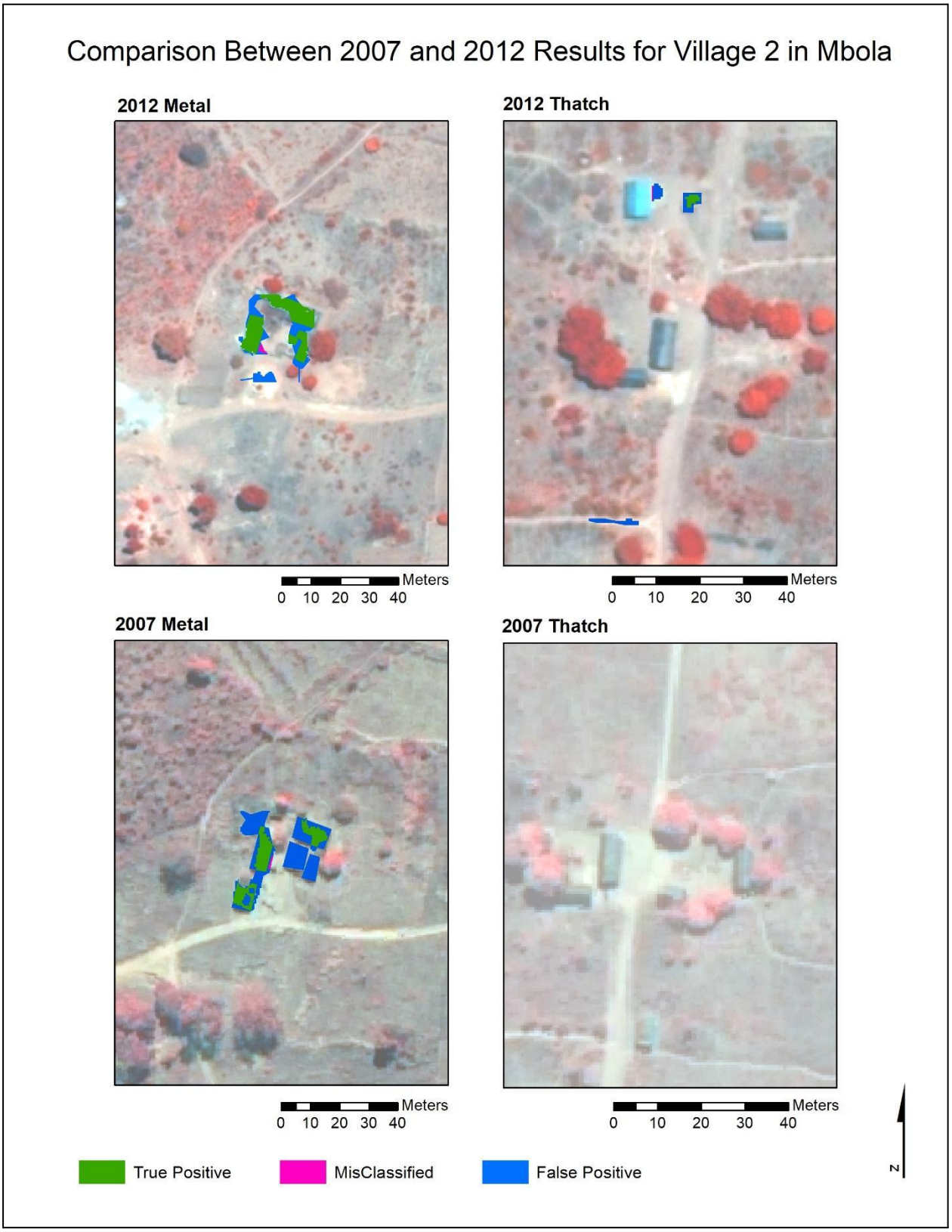
2012 Village 2	*No Roof*	*True Metal (197)*	*True Thatch (19)*	*Producer's Accuracy*
No Roof	N/A	60	11	N/A
Predicted Metal (140)	6	131	3	93.6 %
Predicted Thatch (66)	55	6	5	7.6 %
User's Accuracy	N/A	66.5 %	26.3 %	

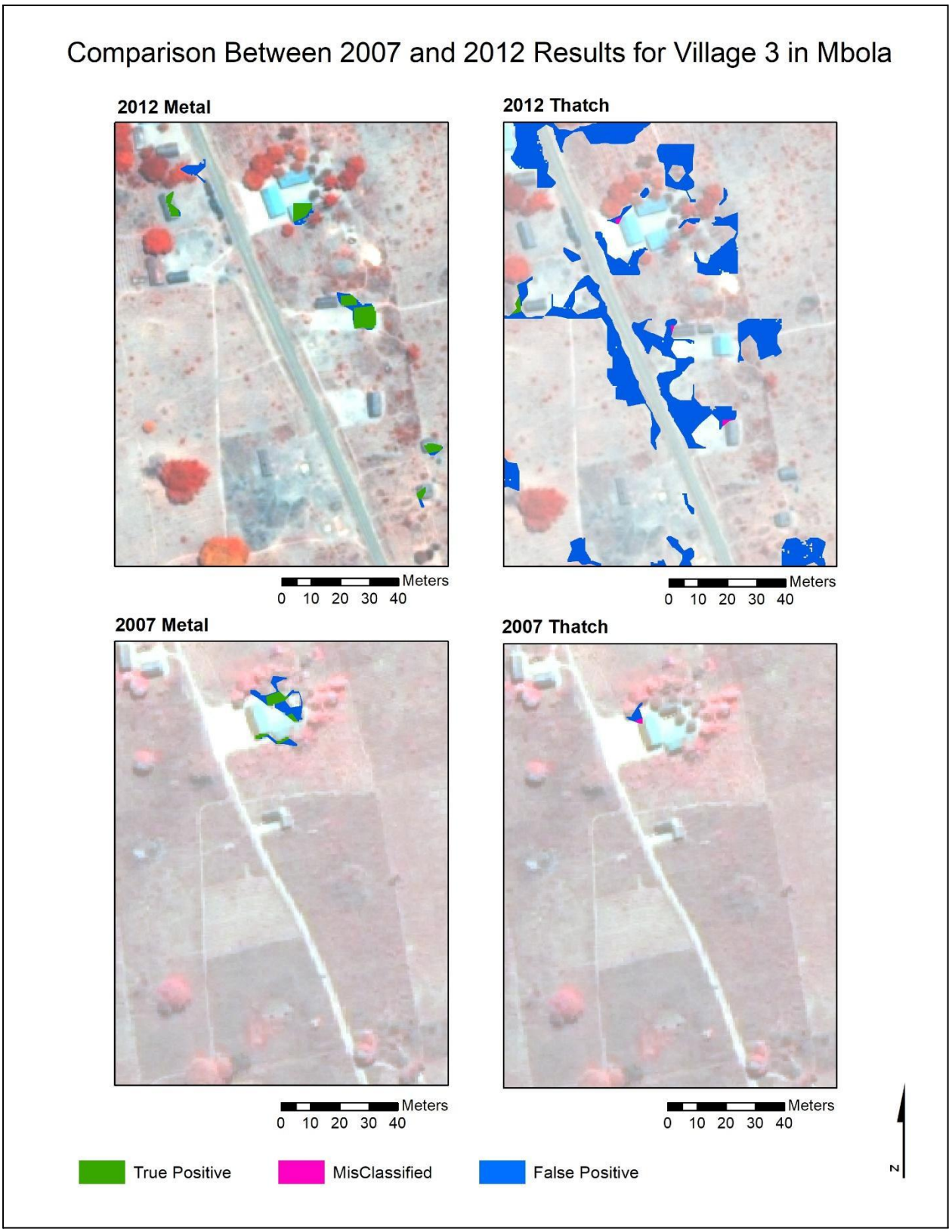
2012 Village 3	*No Roof*	*True Metal (119)*	*True Thatch (14)*	*Producer's Accuracy*
No Roof	N/A	31	0	N/A
Predicted Metal (74)	16	58	0	78.4 %
Predicted Thatch (160)	116	30	14	8.8 %
User's Accuracy	N/A	48.7 %	100 %	

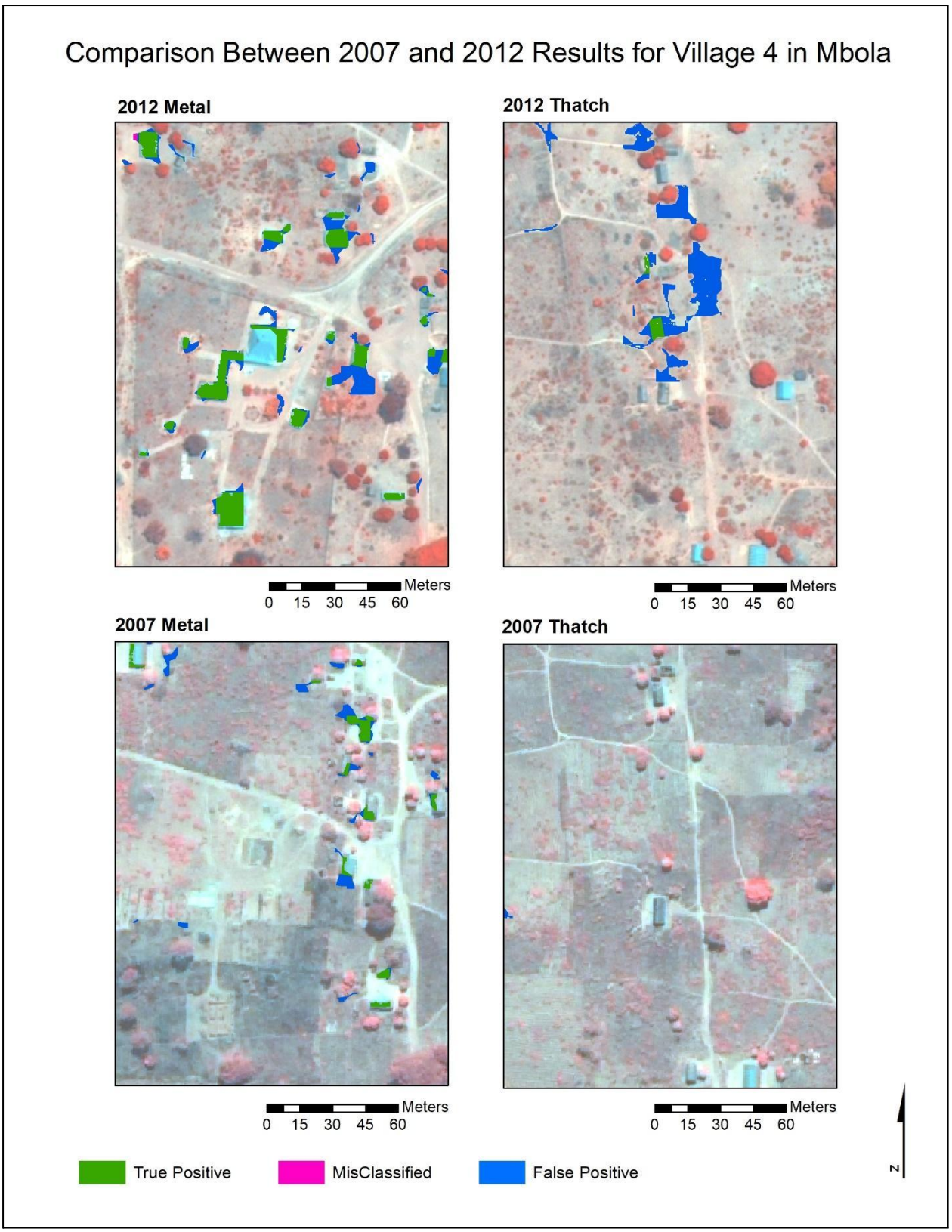
2012 Village 4	*No Roof*	*True Metal (147)*	*True Thatch (19)*	*Producer's Accuracy*
No Roof	N/A	23	8	N/A
Predicted Metal (108)	-2	108	2	100 %
Predicted Thatch (112)	87	16	9	8.0 %
User's Accuracy	N/A	73.5 %	47.4 %	

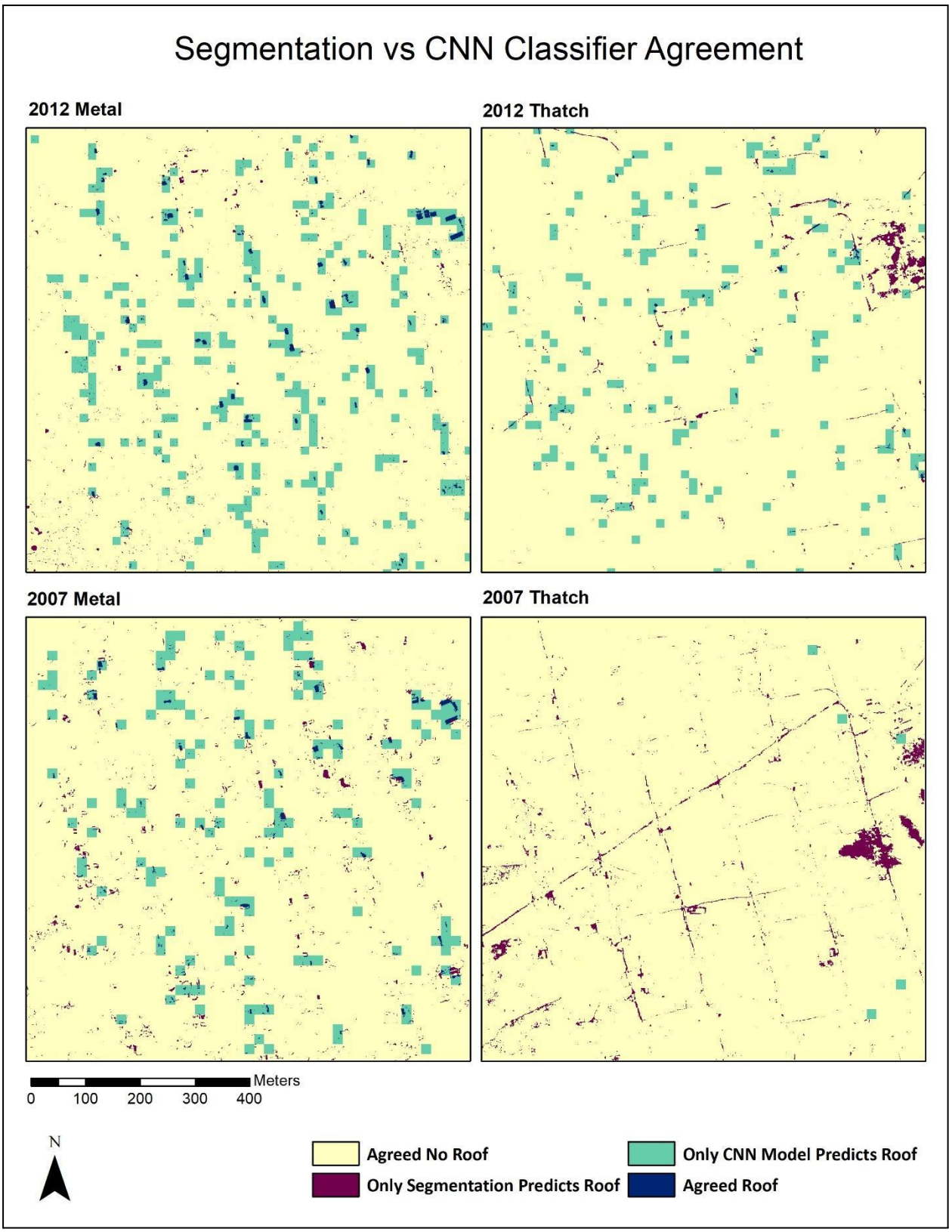
As expected, the model predicted more significantly more metal buildings than thatch, and more buildings in general for the 2012 Mbola dataset than the 2007 dataset.

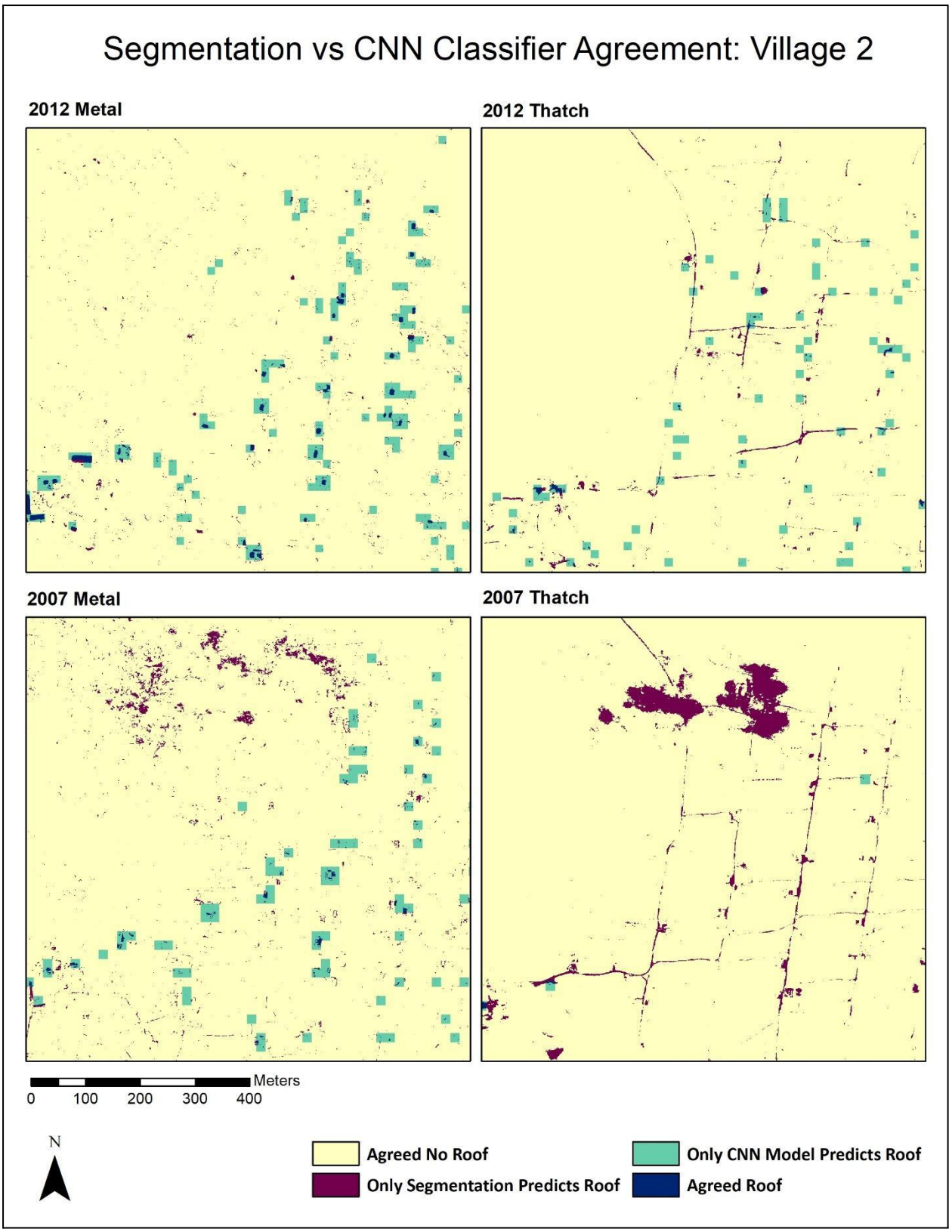
VII: Result Maps

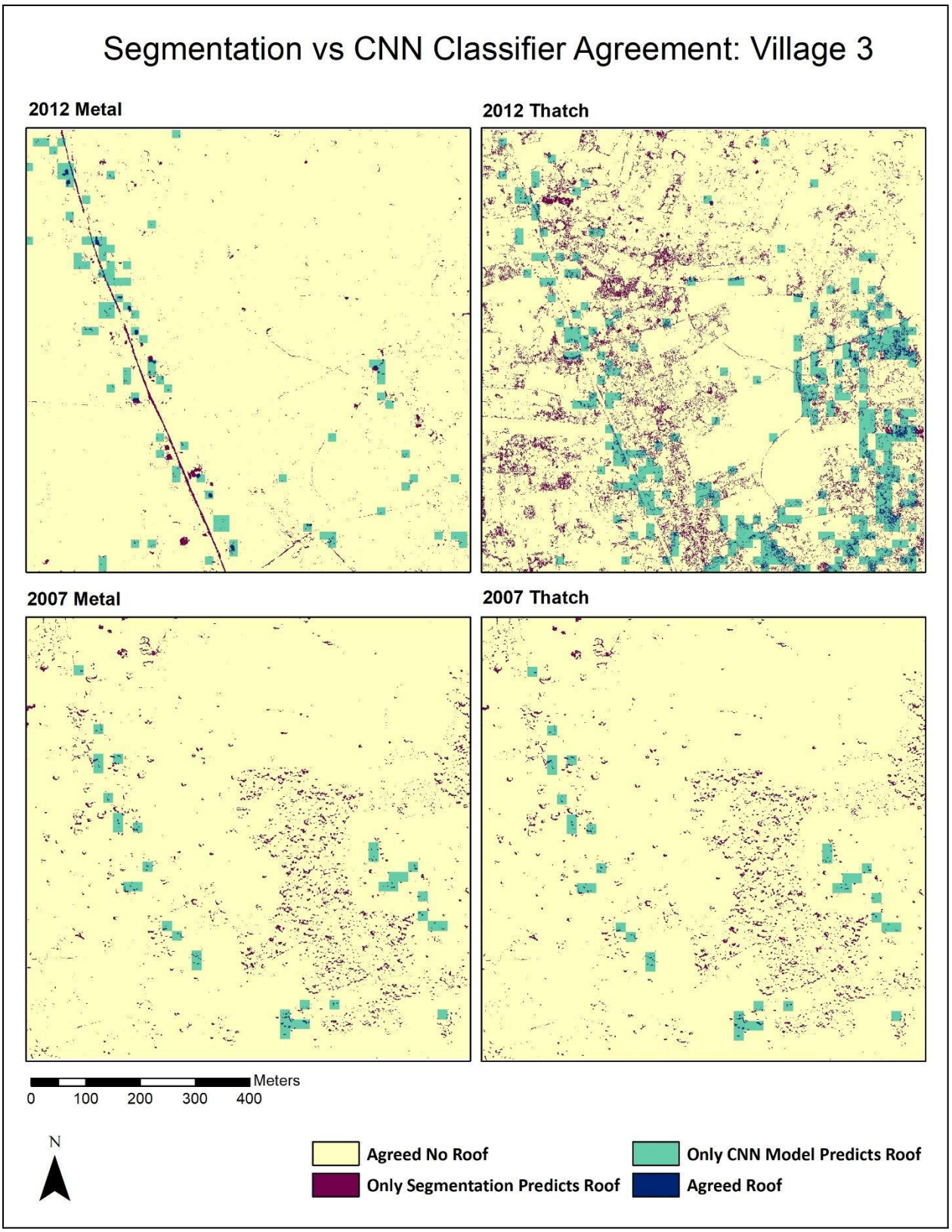


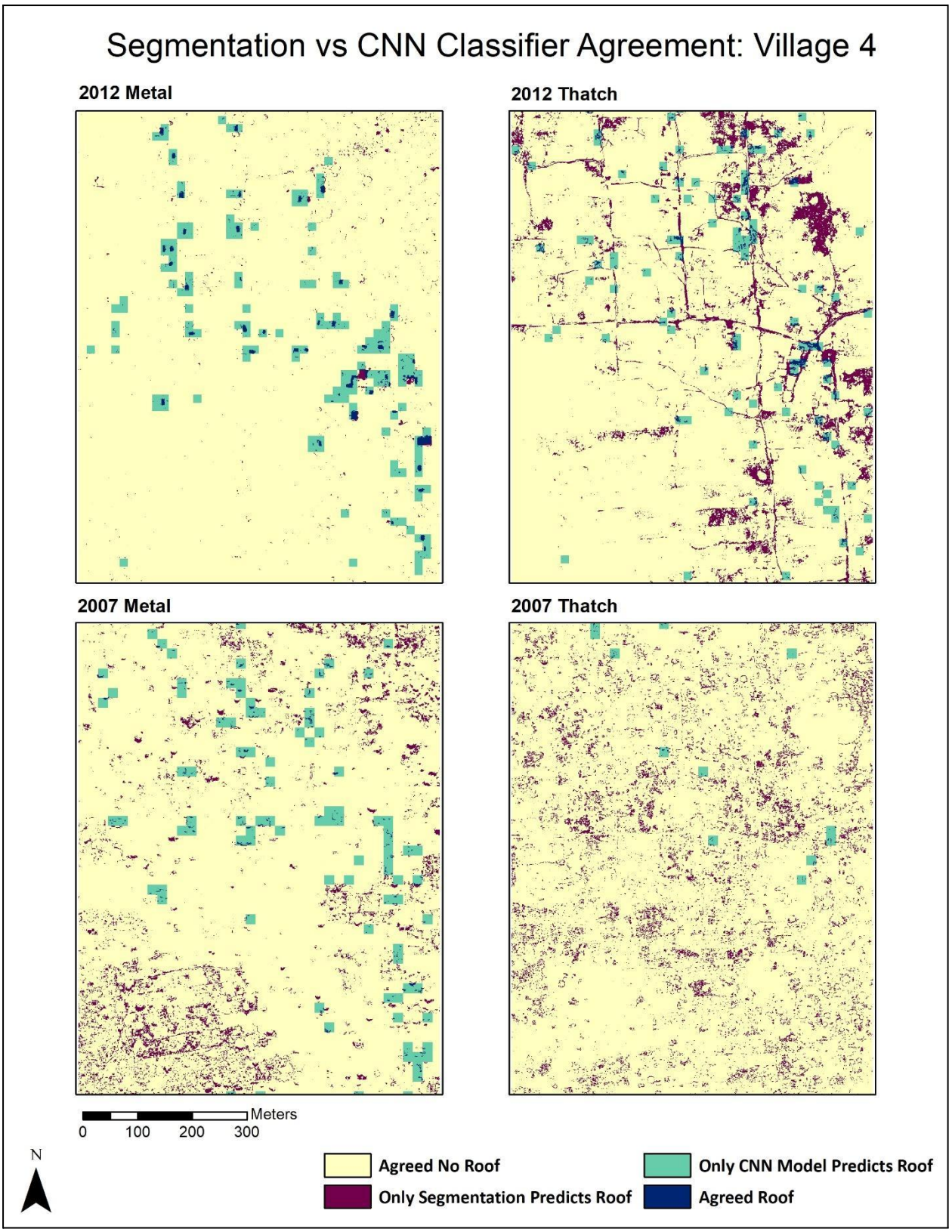

Comparison Between 2007 and 2012 Results for Village 3 in Mbola
2012 Metal
2012 Thatch
Meters
0 10 20 30 40
Meters
0 10 20 30 40
2007 Metal
2007 Thatch
Meters
0 10 20 30 40
Meters
0 10 20 30 40
True Positive
MisClassified
False Positive
N


Comparison Between 2007 and 2012 Results for Village 4 in Mbola
2012 Metal
2012 Thatch
Meters
0 15 30 45 60
Meters
0 15 30 45 60
2007 Metal
2007 Thatch
Meters
0 15 30 45 60
Meters
0 15 30 45 60
True Positive
MisClassified
False Positive
N


Segmentation vs CNN Classifier Agreement
2012 Metal
2012 Thatch
2007 Metal
2007 Thatch
Meters
0
100
200
300
400
N
Agreed No Roof
Only CNN Model Predicts Roof
Only Segmentation Predicts Roof
Agreed Roof


Segmentation vs CNN Classifier Agreement: Village 2
2012 Metal
2012 Thatch
2007 Metal
2007 Thatch
Meters
0 100 200 300 400
N
Agreed No Roof
Only CNN Model Predicts Roof
Only Segmentation Predicts Roof
Agreed Roof


Segmentation vs CNN Classifier Agreement: Village 3
2012 Metal
2012 Thatch
2007 Metal
2007 Thatch
Meters
0
100
200
300
400
N
Agreed No Roof
Only Segmentation Predicts Roof
Only CNN Model Predicts Roof
Agreed Roof


Segmentation vs CNN Classifier Agreement: Village 4
2012 Metal
2012 Thatch
2007 Metal
2007 Thatch
Meters
0 100 200 300
N
Agreed No Roof
Only CNN Model Predicts Roof
Only Segmentation Predicts Roof
Agreed Roof

VIII: Index of Files

Agreement - .tif files of all Agreement outputs
BuildingRoofDetection

- Tanzania
  - Training_data – all training data
  - Segmentation – all segmentation data
  - pansharpenTests
  - pansharpened_Subsets_12
  - pansharpened_Subsets_07
  - Outputs – all post-processing outputs from combined segmentation and CNN
  - Model_Outputs – all output tiffs from CNN model
- Zimbabwe – tiff imagery and ground truth point shapefiles for all roof classes

Figures – all screenshots and map outputs in jpg and png file format
Ground_truth

- Mbola7 – all ground truth shapefiles for 4 Mbola villages 2007
- Mbola12 - all ground truth shapefiles for 4 Mbola villages 2012

Map_mxds – all .mxd files for
Scripts – all .py and .pynb files
Z_Final_Outputs

- Aggregate – all Prediction output shapefiles for Mbola
- MisClassified – all MS output shapefiles for Mbola
- True-Positives – all TP output shapefiles for Mbola
- Zimbabwe – all output shapefiles for Zimbabwe