Northwest Nazarene University

Using Machine Learning to Improve Data Collection of Wildland Fires in Forested Areas

THESIS

Submitted to the Department of Mathematics & Computer Science in

partial fulfillment of the requirements

for the degree of

BACHELOR OF SCIENCE

Kamden Brothers

2021

THESIS

Submitted to the Department of Mathematics & Computer Science in

partial fulfillment of the requirements

for the degree of

BACHELOR OF SCIENCE

By

Kamden Brothers

2021

Using Machine Learning to Improve Data Collection of Wildland Fires

Author: _____

Kamden Brothers

Approved _____

Dr. Dale Hamilton, Department of Mathematics and Computer Science

Advisor

Approved David C Hille _____

David C Hille (Apr 30, 2021 16:00 MDT)

David Hille, Department of Wildlife Biology and Ecology

Second Reader

Approved _____

Dr. Barry Myers, Department of Mathematics and Computer Science

Department Chair

Abstract

Using Machine Learning to Improve Data Collection of Wildland Fires.

Brothers, Kamden (Department of Mathematics and Computer Science),

HAMILTON, DR. DALE (Department of Mathematics and Computer Science)

Using machine learning algorithms on imagery obtained from small unmanned aircraft systems (sUAS) has been an efficient and accurate way to collect data on postfire forests. This effort applies machine learning to obtain useful information about postfire forests. It uses a mask region-based convolutional neural network (MR-CNN) to as well as a support vector machine (SVM) to tree mortality as well as burn extent. Using machine learning helps automated the process while still having accurate data. Having fast and accurate process to calculate the damage done by a fire helps land managers make a quick and calculated response to aid in forest rehabilitation.

Acknowledgements

I would like to thank Dr. Hamilton for this opportunity to be a part of these projects and all that he contributed to them. I would like to thank same for working with me during the summer internship to complete the Tree Mortality project. I would like to thank Jacob Winters for creating the creating the MR-CNN. I would like to thank Dr. Colwell for helping decided how to do the statistical analysis. I would like to thank Cole McCall and Bryn Gautier for creating calibration and validation data.

Contents

Tables and Figures

1. Overview

There were two parts to the research project, both of which involved fires in forested areas, drone imagery, and, machine learning. The first part of the research was "Wildland Fire Tree Mortality Mapping from Hyperspatial Imagery Using Machine Learning" (Hamilton, 2021) and resulted in a published paper on Remote Sensing. This research will be referred to as the Tree Mortality Project. A mask region-based convolutional neural network (MR-CNN) was used along with other methods to calculate the canopy cover of a forested area, creating a hyperspatially derived canopy cover (HDCC) raster. These methods were first done on forested areas that had no recent catastrophic changes. The HDCC raster was compared to LANDFIRE's easily accessible canopy cover layer derived from 30m satellite imagery to determine canopy cover variation between the two methods. The same methods were then used on a postfire forest (LANDFIRE had prefire data while HDCC was postfire). After accounting for bias and variance, the difference between these two layers determined areas of tree mortality. These methods created a highly automated process for calculating tree mortality.

The second part of the research was concerned with burn extent. This research will be referred to as the Tree Island Project. Dr. Hamilton found that using a Support Vector Machine (SVM) on hyperspatial data was an automated, highly accurate way of calculating burn extent (Hamilton, 2021); however, it was also observed that unburnt canopy would sometimes obstruct vision underneath the tree. This would cause the SVM to classify the tree as unburnt even though ground truthing determined that vegetation underneath the tree did burn. This effort attempted to reconcile this problem by locating trees surrounded completely by burn and concluding that underneath the tree was burnt as

well. The research used an SVM to locate burn pixels and an MR-CNN to locate contiguous tree pixels.

2. Background

One of the primary concerns for forest management is rehabilitating a forest after a fire has burnt through it. In extreme circumstances, fires can "leave lands denuded of vegetation and vulnerable to severe erosion and mudslides, which can contaminate municipal water supplies and compromise water quality in streams and Lakes" (Nazzaro, 2006, p. 6). Burn severity and burn extent are useful statistics in helping forest managers appropriately react to a fire's damages. "The mapped fire severity and fire extent can be used to improve both ecological and fuel management" (McCarthy, 2017, p. 64). The Tree Island Project's goal was to make a more accurate map of burn extent by eliminating false negatives, while the Tree Mortality project was focused on burn severity because one of the significant factors of burn severity is tree mortality. "Severity is inherently multifactorial. Some aspects that can be readily quantified are the proportion of foliage consumed or killed, and fire induced tree mortality" (Miquelajauregu, 2016, p. 2). These two projects are concerned with foliage consumption and tree mortality which both help create a more accurate picture of burn severity. They automate the process and allow land managers to acquire data about a fire severity quickly.

3. Methods

3.1 The Tree Mortality Project

The use of imagery from small unmanned aircraft systems (sUAS) has enabled the production of more accurate data about the effects of wildland fire, enabling land managers to make more informed decisions. The ability to detect trees in

hyperspatial imagery enables the calculation of canopy cover. A comparison of hyperspatial post-fire canopy cover and pre-fire canopy cover from sources such as the LANDFIRE project enables the calculation of tree mortality, which is a major indicator of burn severity.

A mask region-based convolutional neural network was trained to classify trees as groups of pixels from a hyperspatial orthomosaic acquired with a small unmanned aircraft system. The tree classification is summarized at 30 m, resulting in a canopy cover raster. A post-fire canopy cover is then compared to LANDFIRE canopy cover preceding the fire, calculating how much the canopy was reduced due to the fire. Canopy reduction allows the mapping of burn severity while also identifying where surface, passive crown, and active crown fire occurred within the burn perimeter… Assessment of canopy reduction mapping on a wildland fire reflects observations made both from ground truthing efforts as well as observations made of the associated hyperspatial sUAS orthomosaic (Hamilton, 2021).

The goal of this project was to automate the process of calculating tree mortality. Figure 1 helps give an overview of the steps taken to convert an orthomosaic (multiple aerial images combined to create one large top-down image) into a map of tree mortality.

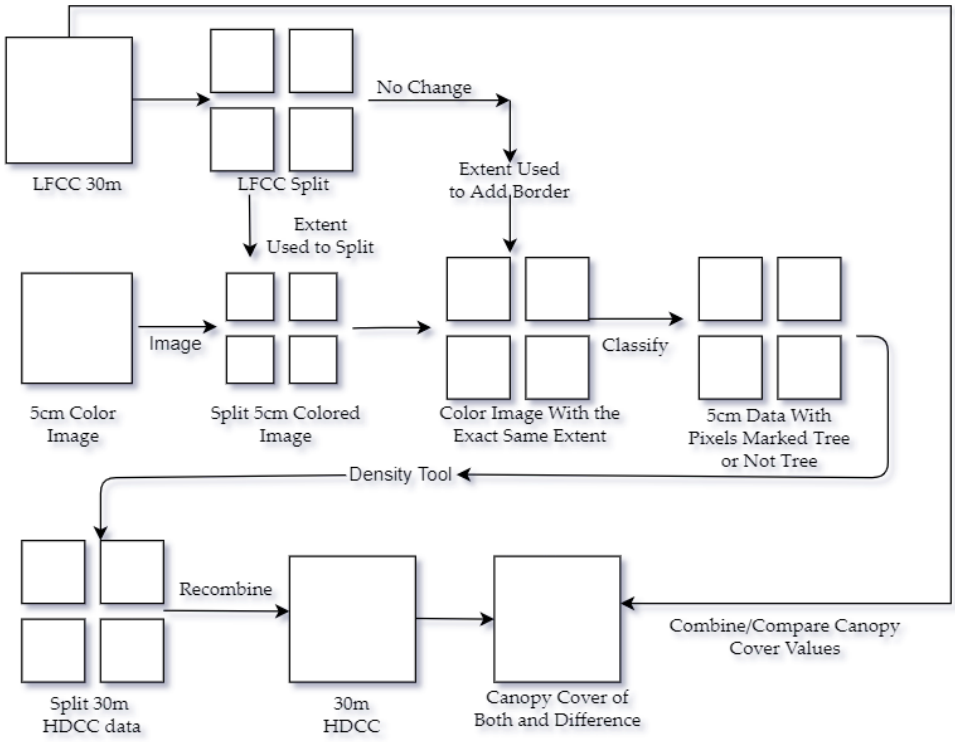


Figure 1 (Hamilton, 2021)

3.1.1 Data Preparation

In Figure 1, LFCC stands for LANDFIRE canopy cover, which is created by Landsat using a support vector machine on 30m data. This data is collected by the United States government and is easily accessible by US citizens. This data is collected across all the USA, which means that any forest fire will have prefire data through LANDFIRE; however, this data is at 30m resolution, affecting accuracy, and is only collected every four years. This data has a low spatial and temporal resolution makes it a nonviable choice for postfire data. Instead, small unmanned aircraft systems (sUAS) were used since they are "...an affordable way to acquire imagery with much higher spatial resolution (sub-decimeter) than was previously available. This hyperspatial imagery

produces homogeneous pixels which are comprised of a single class, resulting in higher mapping accuracy" (Hamilton, 2021).

Ten forests were flown to collect data to compare to the LFCC. These orthomosaics had 5cm resolution and needed to be eventually converted to 30m data to compare to the LFCC. These 5cm orthomosaics were larger than 1 GB, which caused OpenCV (an open-source library used in the Snapper and Density Tool) to have problems. To account for the orthomosaics being too large, they were split up into smaller pieces. First, the LFCC was split into pieces then the 5cm orthomosaic was cropped to these LFCC pieces. Doing it this way caused the edges on the splits of the 5cm data to match the splits of the 30m data. Snapper Tool, created by Nick Hamilton, was used to add a border to the 5cm data to match the 30m data's extent.

3.1.2 Classification and Comparison

The data was classified using a mask region-based convolutional neural network (MR-CNN) trained by Jacob Winters to locate contiguous tree pixels.

> The ... (MR-CNN) is an algorithm for instance segmentation. It detects objects in an image and determines which pixels comprise each object. When classifying an image, MR-CNN begins by applying a convolutional neural network (CNN) to extract features from the image. The features extracted from the image are used as input for a region proposal network, which slides over the feature map and detects regions of interest (RoIs) which are likely to contain objects. Each RoI is evaluated further to determine what type of object, if any, is inside (in this case, tree and non-tree); the object's bounding box; and what pixels inside the bounding box comprise the object (Hamilton, 2021).

The MR-CNN created an output raster with pixels marked tree or not tree. Next, this tree raster was run through the Density Tool, which counted the number of tree pixels in a 30m area and divided it by the total number of pixels. These processes created a 30m hyperspatially derived canopy cover (HDCC) raster with the same extent as the LFCC.

First, an HDCC was calculated for forests where there were no catastrophic changes and compared to the LFCC to calculate the bias and variance. Then the same processes were done on a forest where a fire burned. A null hypothesis was made to find areas where it was known with 95% confidence that there had been a loss in canopy cover.

3.2 The Tree Island Project

Using a support vector machine (SVM) on imagery obtained from small unmanned aircraft systems (sUAS) has been an efficient and fairly accurate method to calculate burn extent. One problem that needs to be addressed is that the vegetation underneath a tree cannot be seen in drone imagery and will be misclassified due to the tree crown which is obscuring the surface vegetation in the image. This research assumes that if a tree is completely surrounded by burnt pixels, then the vegetation under the tree is burnt as well. An SVM was used to locate burn pixels and a mask region-based convolutional neural network (Mask R-CNN) was used to locate clusters of pixels that represent a tree crown. A program was created to located tree pixel clusters which were completely surrounded in the image by pixels that had previously been classified as burned by the SVM and included the cluster of tree crown pixels as being within the extent of the burn. These methods yielded the best results on fires which had a clear distinction between burn pixels, unburned pixels, and tree crowns. These new methods

created will help to calculate a more accurate burn extent from an image to help forest managers rehabilitate burnt forests.

3.2.1 Machine Learning

Both the SVM and MR-CNN are machine learning algorithms that automated the process. A support vector machine takes the input of data for both positive and negative cases. In this case, the data is the RGB band. It then uses these values to create a hyperplane that best splits the data. After the hyperplane is created it is trained and ready to classify an image. The SVM classifies each pixel based on which side of the hyperplane it is on. One side is the positive case the other is the negative case. The other machine learning algorithm used (MR-CNN) was explained in section 3.1.2.

First, an SVM was used to locate burn pixels in a postfire orthomosaic. This created a raster containing burnt and unburnt pixels. Next, the MR-CNN was used on the same orthomosaic to create a raster containing canopy and surface pixels. These rasters were then combined and reclassified so that unburn and surface pixels changed to surface pixels, burn and surface pixels changed to burn pixels, unburn and canopy pixels changed to canopy pixels, and burn and canopy pixels changed to canopy pixels. The reclassification is more clearly stated in Table 1.

Input	**Output**
Unburn + Surface	Surface
Burn + Surface	Burn
Unburn + Canopy	Canopy

Burn + Canopy	Canopy

Table 1 - Reclassification

The reason that burn and canopy pixels were changed to canopy pixels was because the MR-CNN locates groups of pixels which makes sense in the context of each other but the SVM just looks at pixels' values. For this reason, it made more sense to keep the canopy pixels together instead of the burn pixels.

3.2.2 Cleaner

After the Classifiers' outputs were combined there was a raster containing surface, burn, and canopy pixels. This raster had the correct data to be entered into the Tree Island program which locates canopy pixels completely surrounded by burn pixels, however, it was noticed that the canopy pixels did not always go to the edge of trees. These sections of surface pixels would cause problems in locating surrounded trees. To fix this, a Cleaner program was created. This Cleaner locates areas of surface pixels smaller than a specified number of pixels and changes them to burn pixels.

Next, the output of the Cleaner was input into the Tree Island. This program checked each tree to see if it was only touching other tree pixels and burn pixels. If this case was true, then the tree was added to the burn extent.

Determining the threshold to use for the size of surface pixel groups to remove was done using calibration data. Polygons were created around trees that were either a part of the burn extent and not part of the burn extent. These polygons were then compared with the output of the Tree Island program and the threshold that yielded the highest accuracy was chosen.

4. Results

4.1 Results of Tree Mortality Project

Many different results were brought from the project. The first result was a highly accurate, automated process to locate trees in an orthomosaic. The second result was a highly accurate map of canopy cover from drone imagery and the final result is a map of burn severity based on tree mortality.

4.1.1 Locating Trees in an Orthomosaic.

The MR-CNN was run on orthomosaics from ten different forested areas. For each orthomosaic validation data was created by drawing polygons around tree pixels and then drawing polygons around nontree pixels. This group of polygons was then compared to the output of the MR-CNN to calculate the specificity (false positives), sensitivity (correctly identifying tree pixels), and accuracy (chance of correctly identifying any pixel) of the MR-CNN. The results are in Table 2.

	Accuracy	Specificity	Sensitivity
South Placerville	79.5%	98.2%	70.5%
NW Placerville	91.5%	95.1%	88.3%
Edna Creek	99.3%	100.0%	98.8%
South Exp. Forest	89.1%	99.0%	82.2%
North Exp. Forest	89.2%	99.2%	82.6%
East Placerville	95.5%	100.0%	91.6%
West Placerville	92.7%	100.0%	87.3%
Belshazzar	93.4%	93.9%	92.9%
Newstart	82.9%	99.4%	74.8%

Grimes Creek	96.9%	99.6%	94.7%

Table 2 (Hamilton, 2021)

The classifier had 84% sensitivity, 99% specificity, and accuracy of 90%. The MR-CNN was effective at locating the tree and only the tree. It also is automated so larger areas can be classified with little work (Figure 2).

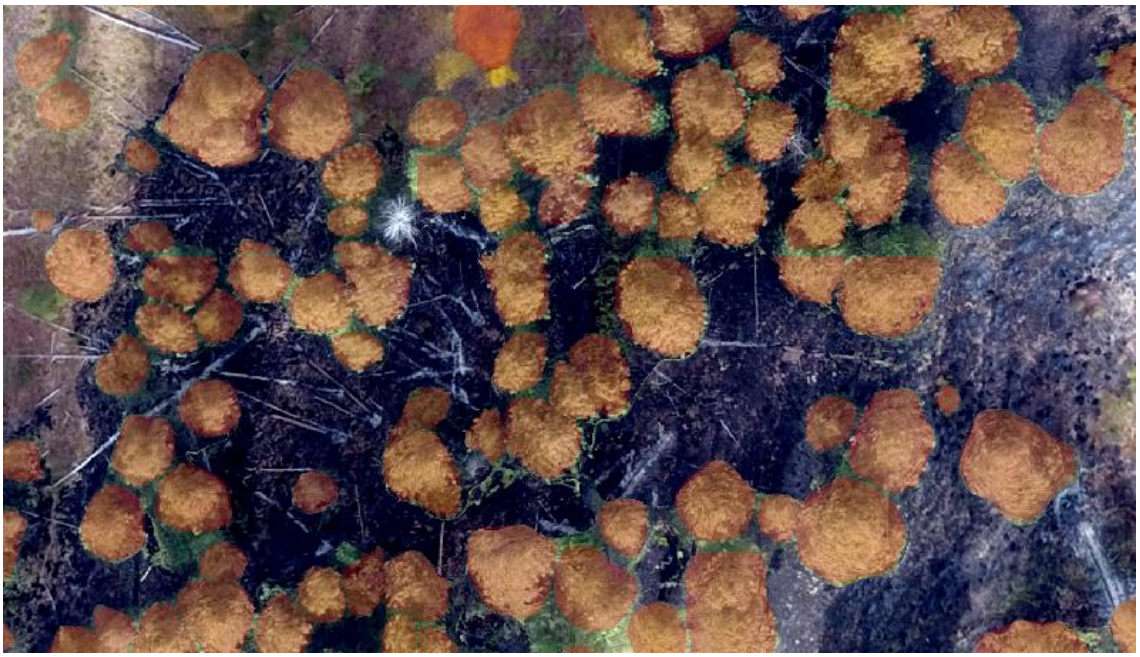


Figure 2 - CottonWood orthomosaic overlayed with MR-CNN output

The MR-CNN does a good job locating the trees; however, its main problem lies in locating where the edges of the trees are. These inconsistencies do not change the map of canopy cover a drastic amount (Hamilton, 2021).

4.1.2 Highly Accurate Map of Canopy Cover

When the MR-CNN data was entered into the Density Tool the result was a map of canopy cover. The output was then adjusted to account for the MR-CNN's bias. The numbers to adjust it came from the sensitivity of 84% and specificity of 99%. These

methods create a highly accurate map of canopy cover (Hamilton, 2021). Figure 3 shows a map of the canopy cover of South Placerville.

Figure 3 - Canopy Cover of South Placerville

4.1.3 Canopy Cover Comparison

I stated the results of the project in "Wildland Fire Tree Mortality Mapping from Hyperspatial Imagery Using Machine Learning"

Ten orthomosaics where there was no fire were compared using the methods stated above. LANDFIRE was over reporting by an average of 2.5% over the HDCC layer. This agrees with what Scott stated, that the "canopy cover values [for LANDFIRE] are too high." The standard deviation for the difference between each pixel was 14. This number is a little high and will hopefully be driven down by future improvements…

The standard deviation and mean collected from the ten non-burnt orthomosaics show that there needs to be at least a change of 27% to know there has been a change with 95% confidence. If the difference is high enough to say there has been a change next the HDCC layer can be looked at to see whether it was a

passive crown fire or an active crown fire. Pixels that did not contain adequate canopy reduction (more than 26%) from comparing pre-fire LFCC to post-fire HDCC are considered to be inconclusive. This would include unburned areas, surface fire, passive crown fire, and active crown fire. If an area had less than 27% canopy cover then we cannot say whether any canopy reduction occurred. This method was used on orthomosaics acquired over portions of the Mesa fire (a 16,000-hectare mixed severity fire located on the Payette National Forest in southern Idaho) to calculate burn intensity. This fire worked well as there were areas of active crown fire, passive crown fire, and surface fire. Figure [4] shows the Southwest area of the Mesa fire which had a lot of Active Crown fire, while Figure [5] the east section which had more passive crown fire and possible surface fire (Hamilton, 2021).

Figure 4. (a) Hyperspatial data after a burn. (b) Types of fire: Red = Active, Yellow = Passive, Black = inconclusive crown fire activity.

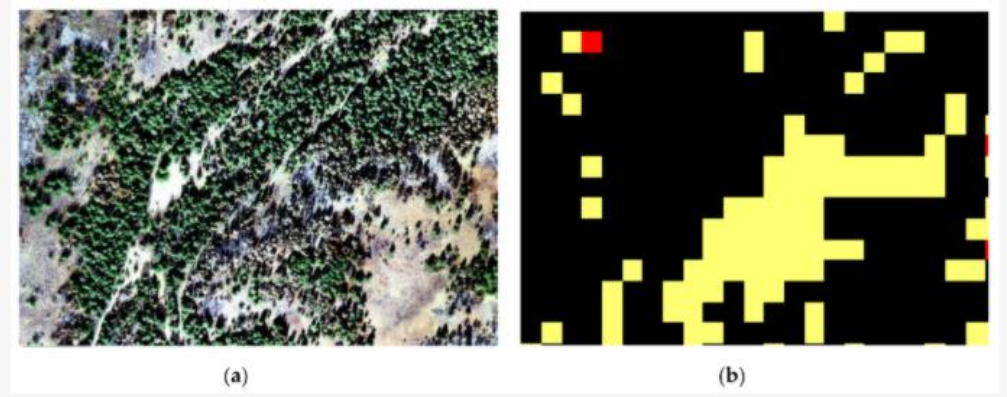
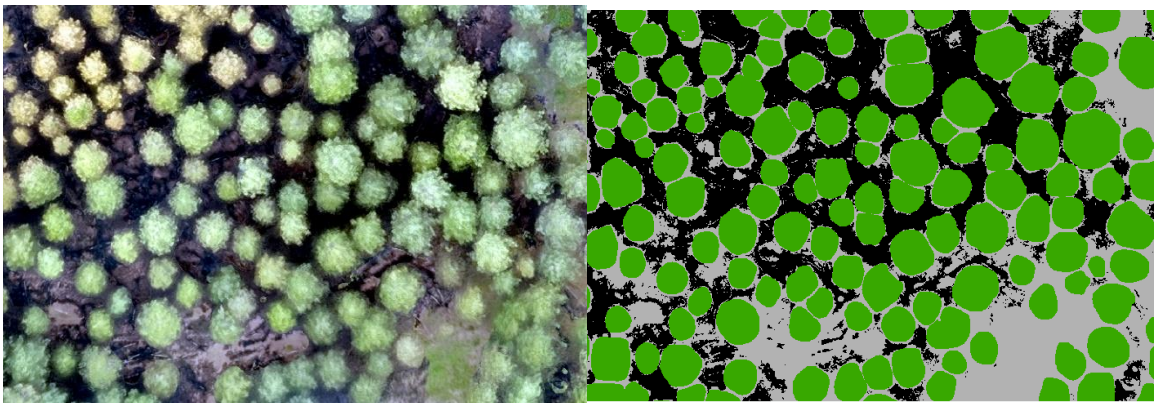


Figure 5. (a) Hyperspatial data after a burn. (b) Types of fire: Red = Active, Yellow = Passive, Black = inconclusive crown fire activity.

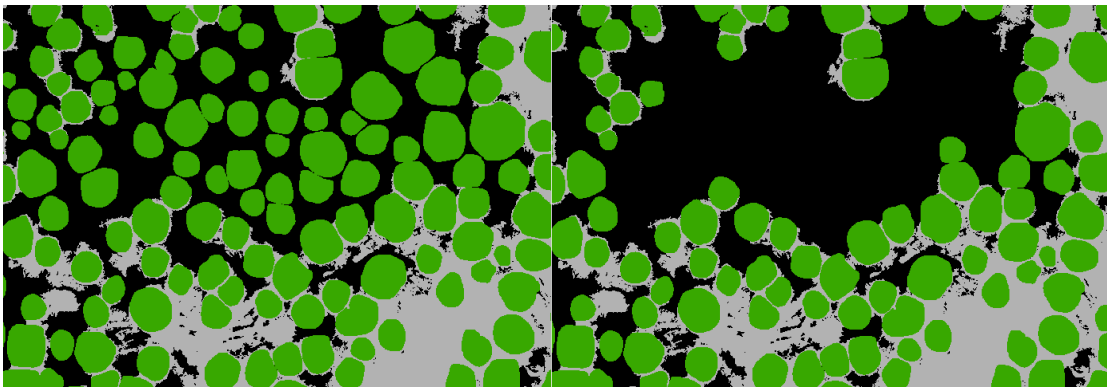
4.2 Results of Tree Island Project

The output of the Tree Island project looks promising. The Hoodoo fire yielded the best results because it did not contain shadows that confuse the SVM. It also had trees that were clearly surrounded by burn pixels. It seemed that higher numbers for the cleaner threshold yielded better results. Figures 6 and 7 show the steps through the process from classification to the cleaner to removing tree islands. The threshold for the cleaner was set to 3200 pixels.



(a) (b)

Figure 6 - Part (a) is an orthomosaic of Hoodoo that was created using drone imagery. Part (b) is the Hoodoo image classified into tree pixels (green), burn pixels (black), and surface pixels (grey)



(a) (b)

Figure 7 - Part (a) is the classified Hoodoo orthomosaic after groups of surface pixels smaller than 3200 are removed. Part (b) is after trees surrounded by burn are removed.

Validation data will be created the same way it was created to test the accuracy of the MR-CNN to find out how much improvement there is in accuracy.

5. Conclusion

Both these projects were ways to apply the skills I have learned to a real-world problem. I used my experience in C++ to create programs useful for the project. My experience in ArcGIS also was used for data preparation and presentation. I also learned how to work with a team to accomplish a task. We used Machine Learning to create highly automated methods that help land managers properly react to the damage done by fires.

Another thing learned through the research project was the work that needed to be done to publish a paper. The most difficult part of it was the editing process. The paper must be to the point where everyone involved including the editors is willing to put their name on it. Lots of people have their reputations at stake when a paper is published.

5.1 Future Work

There are many possibilities for future work to advance the research. Many of them involve the MR-CNN. While the MR-CNN is highly accurate at identifying trees, it can improve. One way it can be improved is at the edges of each tile. The MR-CNN has to split up the image into manageable tiles, however, this causes errors to occur at the edges. One way to fix this would be to create overlapping tiles and only save the inside portion of the tiles. Another bug that needs to be fixed arises when there are too many trees in a tile. This can be fixed by creating smaller tiles. The MR-CNN could be retrained for the Tree Island Project with an emphasis on finding the whole tree.

A paper is being written on the Tree Island Project by Dale and myself that will be published to Remote sensing. This paper was requested for and is a special edition.

References

Hamilton, D. A., Brothers, K. L., Jones, S. D., Colwell, J., & Winters, J. (2021). Wildland Fire Tree Mortality Mapping from Hyperspatial Imagery Using Machine Learning. *Remote Sensing*, *13*(2), 290. doi:10.3390/rs13020290

McCarthy, G., Moon, K., & Smith, L. (2017). Mapping fire severity and fire extent in forest in Victoria for ecological and fuel outcomes. *Ecological Management & Restoration*, *18*(1), 54–65. https://doi.org/10.1111/emr.12242

Miquelajauregui, Y., Cumming, S. G., & Gauthier, S. (2016). Modelling Variable Fire Severity in Boreal Forests: Effects of Fire Intensity and Stand Structure. *PLoS ONE*, *11*(2), 1–24. https://doi.org/10.1371/journal.pone.0150073

Nazzaro, R. M. (2006). Wildland Fire Rehabilitation and Restoration: Forest Service and BLM Could Benefit from Improved Information on Status of Needed Work: GAO-06-670. *GAO Reports*, 1.

Appendix

Cleaner

The cleaner uses openCV to open the images and then stores that data in a Dynamically allocated 2D array.

Input for the Cleaner is %InputImage% %OutputImage% %threshold% %numberofImages%

Threshold is the max size of groups of pixels that will be changed to burn. The cleaner uses a linked list to keep track of pixels. This class is located in TreePixelHT.h. It keeps track of the location of each tree pixel in the group and is also used to iterate through unchecked pixels.

Cleaner.cpp

```
#include <iostream>
#include <opencv2\gapi\plaidml\core.hpp>
#include <opencv2/imgcodecs.hpp>
#include <opencv2\highgui\highgui.hpp>
#include <opencv2\ml\ml.hpp>
#include <cstdio>
#include <ctime>
#include "opencv2\imgproc.hpp"
#include "opencv2\video\background_segm.hpp"
#include "opencv2\video\tracking.hpp"
#include <string>
#include <iomanip>

#include "TreePixelHT.h"
#include <fstream>

using namespace cv;
using namespace std;

const int grass = 0;      //tree number
```

```
const int burn = 1;	//burn number

int Checker(int** arr, int** check, int i, int m, TreePixelHT* uncheckedPix,
TreePixelHT* treeIs, int& checkNum)
{

	if (arr[i][m] == grass)		//is it a grass pixel?
	{

		if (check[i][m] == 0)  //has it been checked?
		{
			//cout << "		" << i << " " << m << endl;
			check[i][m] = checkNum;
			//uncheckedPix->addNode(i, m);	//add to list
			//if (size < thresh)
			treeIs->addNode(i, m);
			return 1;
		}
		else if (check[i][m] != checkNum)
		{
			return -4;
		}
	}
	return 0;
}

bool checkAround(int** arr, int** check, int i, int m, TreePixelHT* uncheckedPix,
TreePixelHT* treeIs, int& checkNum, int rows, int cols, int &size)
{
	//cout << i << " " << m << "\n ";
	int numToAdd = 0;
	bool island = true;
	if (i != 0)
	{
	numToAdd += Checker(arr, check, i - 1, m, uncheckedPix, treeIs, checkNum);

	}//check to the left

	if (m != 0)
	{
```

```
		numToAdd += Checker(arr, check, i, m - 1, uncheckedPix, treeIs,
checkNum);				//check above
	}

	if (i != rows - 1)
	{
		numToAdd += Checker(arr, check, i + 1, m, uncheckedPix, treeIs,
checkNum);			//check to the right
	}

	if (m != cols - 1)
	{
		numToAdd += Checker(arr, check, i, m + 1, uncheckedPix, treeIs,
checkNum);		//check below
	}

	if (numToAdd < 0)
	{
		island = false;
	}
	else
	{
		size += numToAdd;
	}

	return island;
}

int TreeIsland(int** arr, int** check, int cols, int rows, int thresh)
{
	int size;
	int numOfIslands= 0;
	int tenCols, tenRows;
	bool island = true;
	bool helpingBool;
	int checkNum = 0;
	TreePixelHT* uncheckedPix = new TreePixelHT;
	TreePixelHT* treeIs = new TreePixelHT;
	tenCols = rows / 10;
	int diff = rows % 10;
```

```
	for (int i = 0; i < rows; i++)			//i is rows
	{
		if ((i-diff) % tenCols == 0)
		{
			cout << setw(2) << 100 * (i -diff) / (double)(rows-diff) << "% done | ";
		}
		for (int m = 0; m < cols; m++)			//m is columns
		{
			size = 0;
			if (check[i][m] == 0)
			{
				check[i][m] = checkNum + 1;

				if (arr[i][m] == grass) //unchecked tree pixel. Check for island
				{
					checkNum++;
					size++;
					island = checkAround(arr, check, i, m, uncheckedPix, treeIs, checkNum, rows, cols, size);
					treeIs->addNode(i, m);


					TreePixel* place;
					place = treeIs->Head;
					while (place != nullptr && size < thresh*2)
					{
						int r = place->row;
						int c = place->column;

						helpingBool = checkAround(arr, check, r, c, uncheckedPix, treeIs, checkNum, rows, cols, size);
						if (helpingBool == false)
						{
							island = false;
						}
						place = place->next;
					}
```

```
                                        if (size < thresh && island)
                                        {
                                                numOfIslands++;
                                                size = 0;
                                                while (treeIs->Tail != nullptr)
                                                {
                                                        TreePixel* tester = treeIs->Tail;
                                                        size++;
                                                        int r = treeIs->Tail->row;
                                                        int c = treeIs->Tail->column;
                                                        treeIs->deleteTail();
        //remove end of list

                                                        arr[r][c] = burn;


                                                }
                                                //cout << "size: " << size << endl;
                                        }
                                        else
                                        {
                                                while (treeIs->Tail != nullptr)
                                                {
                                                        treeIs->deleteTail();
                //delete treeIs list
                                                }
                                        }
                                }
                        }
                }
        }
        cout << endl << numOfIslands << endl;
        return 1;
}

int main(int argc, char* argv[])
{
        int help;
        int thresh;
        Mat image;
```

```
	clock_t timer;		//Keeps track of time
	double duration;
	string inputBaseName, inputWithNum, outputBaseName, outputWithNum;
	int numOfImages = 1;

	if (argc > 3)
	{
		inputBaseName = inputWithNum = argv[1];
		outputBaseName = outputWithNum = argv[2];
		thresh = stoi(argv[3]);
		if (argc > 4 && argv[4] != "1")
		{
			numOfImages = stoi(argv[4]);
		}
	}
	else
	{
		cout << "no command line.";
		return 1;
	}

	for (int j = 0; j < numOfImages; j++)
	{
		if (numOfImages != 1)
		{
			inputWithNum = "";
			outputWithNum = "";
			cout << "Using base name " << argv[1] << endl << "for " <<
numOfImages << " images\n";
			int inlength = inputBaseName.length();
			int outlength = outputBaseName.length();
			for (int k = 0; k < inlength; k++)
			{
				inputWithNum += inputBaseName[k];

				if (k == inlength - 5) //before .tif
				{
					inputWithNum += to_string(j);
				}
```

```
                        }
                        for (int k = 0; k < outlength; k++)
                        {
                                outputWithNum += outputBaseName[k];

                                if (k == outlength - 5) //before .tif
                                {
                                        outputWithNum += to_string(j);
                                }

                        }
                        //cout << endl << inputWithNum;
                        //cin >> inputWithNum;

                }
                cout << "Opening " << inputWithNum << endl;

                image = imread(inputWithNum, IMREAD_UNCHANGED);         //open
image

                if (image.empty())
        //Check if image opened properly
                {
                        cout << "\nCould not open " << inputWithNum;
                        return 1;
                }

                int rows;
                int columns;

                rows = image.rows;
                columns = image.cols;
                timer = clock();
        //start timer
                cout << "\n\n" << rows << " rows and " << columns << " columns \n";

                int** arr;
                int** check;
                check = new int* [rows];                //dynamically allocate 2d array to
keep track of checked pixels
```

```
		arr = new int* [rows];			//dynamically allocate 2d array that
will hold tree pixels
		for (int m = 0; m < rows; m++)
		{
			check[m] = new int[columns];
			arr[m] = new int[columns];
		}

		for (int i = 0; i < rows; i++)
		{
			for (int m = 0; m < columns; m++)
			{
				check[i][m] = 0;
	//set checks to 0
				arr[i][m] = (int)image.at<ushort>(i, m);		//input
mat into an array

			}
		}


		TreeIsland(arr, check, columns, rows, thresh);
	//check for tree islands

		duration = ((double)clock() - (double)timer) /
(double)CLOCKS_PER_SEC;		//end timer

		cout << endl << duration << "s to find all grass pixels < " << thresh <<
"\n\nSaving image to " << outputWithNum << "\n\n";		//output time

		for (int i = 0; i < rows; i++)
		{
			for (int m = 0; m < columns; m++)
			{
				image.at<ushort>(i, m) = (ushort)arr[i][m];
	//input array into the mat
			}
		}

		for (int i = 0; i < rows; i++)		//delete tables
```

```
                {
                        delete[] check[i];
                        delete[] arr[i];
                }
                delete[] check;
                delete[] arr;

                imwrite(outputWithNum, image);      //save image
        }
}
```

TreeIsland Program

TreeIsland.cpp

```
#include <iostream>

#include <opencv2\gapi\plaidml\core.hpp>

#include <opencv2/imgcodecs.hpp>

#include <opencv2\highgui\highgui.hpp>

#include <opencv2\ml\ml.hpp>

#include <cstdio>

#include <ctime>

#include "opencv2\imgproc.hpp"

#include "opencv2\video\background_segm.hpp"

#include "opencv2\video\tracking.hpp"


#include <string>

#include "TreePixelHT.h"
```

```
#include <fstream>


using namespace cv;

using namespace std;


const int tree = 4;        //tree number

const int burn = 1;        //burn number


bool Checker(int** arr, int** check, int i, int m, TreePixelHT* uncheckedPix,
TreePixelHT* treeIs, int &checkNum)

{


        if (arr[i][m] == tree)              //is it a tree pixel?

        {


                if (check[i][m] == 0)   //has it been checked?

                {

                        //cout << "          " << i << " " << m << endl;

                        check[i][m] = checkNum;

                        uncheckedPix->addNode(i, m);         //add to list

                        treeIs->addNode(i, m);

                        return true;                         //still could be a tree
```

```
                }
                else if (check[i][m] != checkNum)     //already checked and not a tree
island
                {
                        //cout << 1;
                        return false;                       //not a tree Island
                }
                else                                        //already checked
                {
                        //cout << 2;
                        return true;
                }
        }
        else if (arr[i][m] == burn)                 // surrounded by a non burn or tree pixel
        {
                //cout << 3;
                return true;
        }
        else
        {
                //cout << 4;
                return false;
```

```
	}

}


bool checkAround(int** arr, int** check, int i, int m, TreePixelHT* uncheckedPix,
TreePixelHT* treeIs, int &checkNum)

{

	//cout << i << " " << m << "\n ";

	bool island = true;

	island = Checker(arr, check, i - 1, m, uncheckedPix, treeIs, checkNum);
			//check to the left

	if (island)

	{

		//Check above

		island = Checker(arr, check, i, m - 1, uncheckedPix, treeIs, checkNum);
			//check above

		if (island)

		{

			//Check Right

			island = Checker(arr, check, i + 1, m, uncheckedPix, treeIs,
checkNum);			//check to the right

			if (island)

			{

				//Check Below
```

```
                                island = Checker(arr, check, i, m + 1, uncheckedPix, treeIs,
checkNum);              //check below

                                }

                        }

                }

                return island;

}


int TreeIsland(int** arr, int** check, int cols, int rows)

{

                int size;

                bool island = true;

                int checkNum = 0;

                TreePixelHT* uncheckedPix = new TreePixelHT;

                TreePixelHT* treeIs = new TreePixelHT;


                for (int i = 0; i < rows; i++)                  //i is rows

                {

                        //cout << endl << i << endl;

                        for (int m = 0; m < cols; m++)                  //m is columns

                        {

                                island = true;
```

```
                    if (check[i][m] == 0)
                    {
                        check[i][m] = checkNum + 1;

                        if (arr[i][m] == tree)    //unchecked tree pixel. Check for
island
                        {
                            if (i != 0 && i != rows - 1 && m != 0 && m != cols
- 1) //not an edge pixel
                            {
                                checkNum++;
                                island = checkAround(arr, check, i, m,
uncheckedPix, treeIs, checkNum);
                                treeIs->addNode(i, m);
                            }
                            else
                            {
                                island = false;
                            }

                            TreePixel *place;
                            place = uncheckedPix->Head;
                            while (place != nullptr && island)
```

```
{
        //cout << endl << endl;
        int r = place->row;
        int c = place->column;

        if (r != 0 && r != rows - 1 && c != 0 && c
!= cols - 1) //not an edge pixel
        {
                island = checkAround(arr, check, r,
c, uncheckedPix, treeIs, checkNum);
                place = place->next;
        }
        else
        {
                island = false;
        }
}
if (island)
{
        cout << "island found ";
        size = 0;
        while (treeIs->Tail != nullptr)
        {
```

```
                    TreePixel *tester = treeIs->Tail;
                    //cout << endl << endl;
                    size++;
                    int r = treeIs->Tail->row;
                    int c = treeIs->Tail->column;
                    treeIs->deleteTail();
//remove end of list

                    arr[r][c] = burn;
//Tree is burnt

                }
                cout << "size: " << size << endl;
            }
            else
            {
                while (uncheckedPix->Tail != nullptr)
//clean up
                {
                    uncheckedPix->deleteTail();
    //delete uncheckedPix list
                }
                while (treeIs->Tail != nullptr)
                {
```

```
                                                    treeIs->deleteTail();
                //delete treeIs list
                                                }
                                            }
                                        }
                                    }
                                }
            }
            return 1;
}

int main(int argc, char* argv[])
{
            int help;
            Mat image;
            string inputBaseName;
            string outputBaseName;
            string inputWithNum;
            string outputWithNum;
            int numOfImages;

            clock_t timer;              //Keeps track of time
```

```
double duration;


if (argc > 2)
{
	inputBaseName = inputWithNum = argv[1];
	outputBaseName = outputWithNum = argv[2];
	if (argc > 3 && argv[3] != "1")
	{
		cout << argv[3] << endl << endl;
		numOfImages = stoi(argv[3]);
	}
	else
	{
		numOfImages = 1;
	}


}
else
{
	cout << "no command line.";
	return 1;
}
```

```
	for (int j = 0; j < numOfImages; j++)
	{
		if (numOfImages != 1)
		{
			inputWithNum = "";
			outputWithNum = "";
			cout << "Using base name " << argv[1] << endl << "for " << numOfImages << " images\n";
			int inlength = inputBaseName.length();
			int outlength = outputBaseName.length();
			for (int k = 0; k < inlength; k++)
			{
				inputWithNum += inputBaseName[k];

				if (k == inlength - 5) //before .tif
				{
					inputWithNum += to_string(j);
				}

			}
			for (int k = 0; k < outlength; k++)
```

```
                {
                        outputWithNum += outputBaseName[k];

                        if (k == outlength - 5) //before .tif
                        {
                                outputWithNum += to_string(j);
                        }

                }
                //cout << endl << inputWithNum;
                //cin >> inputWithNum;

        }
        cout << "Opening " << inputWithNum << endl;

        image = imread(inputWithNum, IMREAD_UNCHANGED);         //open
image

        if (image.empty())
    //Check if image opened properly
        {
                cout << "\nCould not open " << inputWithNum;
                return 1;
```

```
}

int rows;
int columns;

rows = image.rows;
columns = image.cols;

cout << "\n\n" << rows << " rows and " << columns << " columns \n";

int** arr;
int** check;
check = new int* [rows];          //dynamically allocate 2d array to keep track of checked pixels
arr = new int* [rows];            //dynamically allocate 2d array that will hold tree pixels
for (int m = 0; m < rows; m++)
{
    check[m] = new int[columns];
    arr[m] = new int[columns];
}

for (int i = 0; i < rows; i++)
```

```
		{
			for (int m = 0; m < columns; m++)
			{
				check[i][m] = 0;
	//set checks to 0
				arr[i][m] = (int)image.at<ushort>(i, m);		//input mat into an array

			}
		}

		timer = clock();
	//start timer

		TreeIsland(arr, check, columns, rows);		//check for tree islands

		duration = ((double)clock() - (double)timer) / (double)CLOCKS_PER_SEC;	//end timer

		cout << endl << duration << "s to find all islands\n\nSaving image " << outputWithNum << "\n\n";	//output time

		for (int i = 0; i < rows; i++)
		{
```

```
                for (int m = 0; m < columns; m++)
                {
                    image.at<ushort>(i, m) = (ushort)arr[i][m];
        //input array into the mat
                }
            }

            imwrite(outputWithNum, image);      //save image

            for (int i = 0; i < rows; i++)              //delete tables
            {
                delete[] check[i];
                delete[] arr[i];
            }
            delete[] check;
            delete[] arr;
        }
}
```

TreePixelHT.h

```
#pragma once
#include "TreePixel.h"
```

```
class TreePixelHT
{
public:
	TreePixel* Head;
	TreePixel* Tail;

	TreePixelHT()
	{
		Head = Tail = nullptr;
	}

	void addNode(int, int);
	void deleteTail();
};
```

TreePixelHT.cpp

```
#include "TreePixelHT.h"
#include <iostream>
using namespace std;

void TreePixelHT::addNode(int first, int second)
{
```

```
        TreePixel* ptr;
        ptr = new TreePixel;            //making new node

        ptr->row = first;                       //adding data
        ptr->column = second;

        if (this->Head == nullptr)      //empty list
        {
                this->Head = this->Tail = ptr;//set head and tail
                return;
        }
        else
        {
                TreePixel* help;
                ptr->prev = help = this->Tail;          //add node to the beginning
                help->next = this->Tail = ptr;
        }
}


void TreePixelHT::deleteTail()                  //remove node from the tail
```

```
{
	TreePixel* ptr = this->Tail;

	if (ptr->prev == nullptr)
	{
		delete ptr;
		this->Tail = this->Head = nullptr;
		return; //empty list
	}
	else
	{
		this->Tail = ptr->prev;
		delete ptr;
		return; //still items in list
	}
}
```

TreePixel.h

```
#pragma once
class TreePixel
{
public:
```

```
        TreePixel* next;
        TreePixel* prev;
        int column;
        int row;

        TreePixel ()
        {
                next = prev = nullptr;
                row = column = -1;
        }
};
```

KamdenFinalSeniorThesis

Final Audit Report 2021-05-04

Created:	2021-04-29
By:	Kamden Brothers (kbrothers@nnu.edu)
Status:	Signed
Transaction ID:	CBJCHBCAABAA4EwQZ-lpj_-_YD90mcOSGTH-asSv6vwQ

"KamdenFinalSeniorThesis" History

Document created by Kamden Brothers (kbrothers@nnu.edu)
2021-04-29 - 11:03:36 PM GMT- IP address: 198.60.209.202

Document emailed to David C Hille (dhille@nnu.edu) for signature
2021-04-29 - 11:06:44 PM GMT

Email viewed by David C Hille (dhille@nnu.edu)
2021-04-30 - 1:47:55 AM GMT- IP address: 66.249.84.95

Document e-signed by David C Hille (dhille@nnu.edu)
Signature Date: 2021-04-30 - 10:00:51 PM GMT - Time Source: server- IP address: 198.60.209.90

Document emailed to Dale Hamilton (dhamilton@nnu.edu) for signature
2021-04-30 - 10:00:55 PM GMT

Email viewed by Dale Hamilton (dhamilton@nnu.edu)
2021-05-02 - 4:27:43 AM GMT- IP address: 66.249.84.91

Email viewed by Dale Hamilton (dhamilton@nnu.edu)
2021-05-04 - 2:49:08 AM GMT- IP address: 66.249.84.217

Document e-signed by Dale Hamilton (dhamilton@nnu.edu)
Signature Date: 2021-05-04 - 2:55:16 AM GMT - Time Source: server- IP address: 67.60.216.39

Document emailed to Barry L Myers (blmyers@nnu.edu) for signature
2021-05-04 - 2:55:19 AM GMT

Email viewed by Barry L Myers (blmyers@nnu.edu)
2021-05-04 - 3:12:39 PM GMT- IP address: 66.249.84.205

Document e-signed by Barry L Myers (blmyers@nnu.edu)
Signature Date: 2021-05-04 - 3:13:29 PM GMT - Time Source: server- IP address: 174.27.93.244

Agreement completed.

2021-05-04 - 3:13:29 PM GMT