NORTHWEST NAZARENE UNIVERSITY

Classifying wildland fire severity on Landsat imagery using Machine Learning trained by hyperspatial imagery

THESIS
Submitted to the Department of Mathematics and Computer Science
in partial fulfillment of the requirements
for the degree of
BACHELOR OF SCIENCE

Nicholas A. Hamilton

THESIS
Submitted to the Department of Mathematics and Computer Science
in partial fulfillment of the requirements
for the degree of
BACHELOR OF SCIENCE

Nicholas A. Hamilton
2019

Classifying wildland fire severity on Landsat imagery using Machine Learning trained by hyperspatial imagery

Author: [illegible]
Nicholas A. Hamilton

Approved: [illegible]
Dale A. Hamilton, Ph.D., Faculty Advisor
Department of Mathematics and Computer Science,

Approved: [illegible]
Benjamin Gall, Ph.D., Second Reader
Department of Kinesiology

Approved: Barry Myers
Barry L. Myers, Ph.D., Chair,
Department of Mathematics & Computer Science

Abstract

Classifying wildland fire severity on Landsat imagery using Machine Learning trained by hyperspatial imagery

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

Many different machine-learning algorithms have previously been used to map wildland fire effects using satellite imagery from the Landsat satellites with 30-meter spatial resolution. Small-unmanned aircraft systems (sUAS) can capture images with five-centimeter (hyperspatial) resolution. Consequently, the amount of data needing to be stored and analyzed significantly increased. There is a need for more tools that focus on extracting actionable knowledge from hyperspatial imagery and providing timely information for management of wildland fires. This analysis shows that the accurate mapping of fire effects from hyperspatial imagery increased from 56.62% to 93.16% for Burn Extent and 28.4% to 95.94% for Biomass Consumption. The classifier developed to do this analysis uses a support vector machine (SVM) to determine the burn severity by classifying image pixels into canopy crown, surface vegetation, white ash, and black ash.

The use of sUAS to map burn severity creates another problem. The flight time of sUAS allows them to have the capability only to map small fires. Classifications were modified to utilize machine-learning algorithms. Images, obtained from Landsat, are analyzed using the new classification. Implementing the new classification allows, not only small fires but, large fires to be modeled as well.

Acknowledgments

There are many people who I would like to thank. First is my father and professor Dr. Hamilton. He helped me through difficult problems and encouraged me the entire way. As well as the opportunities to coauthor academically published papers. I would also like to thank Dr. Myers for helping me with constant help and comments. There were many students who came before me that have laid the foundation for this project, they deserve my thanks as well. Finally, I would like to thank all our sponsors and collaborators. These include: NASA Space Grant Consortium, NASA Established Program to Stimulate Competitive Research (EPSCOR), USDA Forest Service Boise & Payette National Forest (NF), Rocky Mountain Research Station, the Bureau of Land Management (BLM) Idaho State Office as well as an Institutional Development Award (IDeA) from the National Institute of General Medical Sciences of the National Institutes of Health under Grant #P20GM103408.

Table of Contents

List of Figures

1. Overview

The goal of my thesis is to determine whether algorithms developed as part of the Fire Monitoring and Assessment Platforms (FireMAP) research effort can efficiently be used to map fire extent on Landsat imagery while using small unmanned aircraft systems (sUAS) imagery as training data allowing the mapping of not only small fires but large wildland fires that exceed the flight extent of an sUAS as well. The FireMAP algorithms needed several modifications to accomplish our goals. To summarize the modifications, edges of the image needed to be padded with additional pixels so the image extent lined up with the pixels in the Landsat image. Adding the border allows the two images' pixels to be lined up. Even though there has been no validation, the initial results are promising.

2. Background

One of the major purposes of technology is to aid humans by making their life easier, aiding them in decision making and helping keep them safe. FireMAP is an ongoing research project conducted by Northwest Nazarene University (NNU) Department of Math & Computer Science. Collaborators include USDA Forest Service Boise National Forest, and Payette National Forest, Rocky Mountain Research Station as well as the Bureau of Land Management (BLM) Idaho State Office. The results of this project helps Burned Area Emergency Response (BAER) Teams, people who rehabilitate a landscape after a fire (WO Staff Program - Burned Area Emergency Response BAER. /n.d.). It should make their jobs easier because they will no longer need to walk the perimeter of the burn and map the extent of it by hand. BAER Teams will receive a complete map of the burn that provides them with not only the extent of the fire but also where it burned most severely. These maps will be temporally sensitive and high spatial resolution, 5cm. The new information will also help them with decision making. Perhaps more importantly, it will keep humans safe. Even an extinguished fire is a dangerous environment for humans to work. Structurally unsound trees are known to fall upon and injure or kill unsuspecting people.

2.1. Spatial Resolution Experiment

The purpose of the spatial experiment, the predecesor to this project, was to determine if a map of burn severity has higher accuracy using hyperspatial imagery with a spatial resolution of 5 centimeters (5cm) than is possible with medium resolution color imagery with a spatial resolution of 30 meters (30m). The decrease in spatial resolution from 5cm happened by using fuzzy logic to convert sUAS imagery to the same spatial resolution as Landsat. Fuzzy logic differs from binary logic because it is not black and white like binary logic is. Fuzzy logic can have some grey areas. It can transition from 0 to 1 over a range of values. Sometimes it can have a value of 0.501. In binary logic then the value would be 1. If the fuzzy input had a value was 0.499, only 0.002 percent less, then the binary value would be 0. In this example, even though the fuzzy values are .002 different, the binary values would be 0 and 1 (Hamilton, Hamilton, & Myers, 2019).

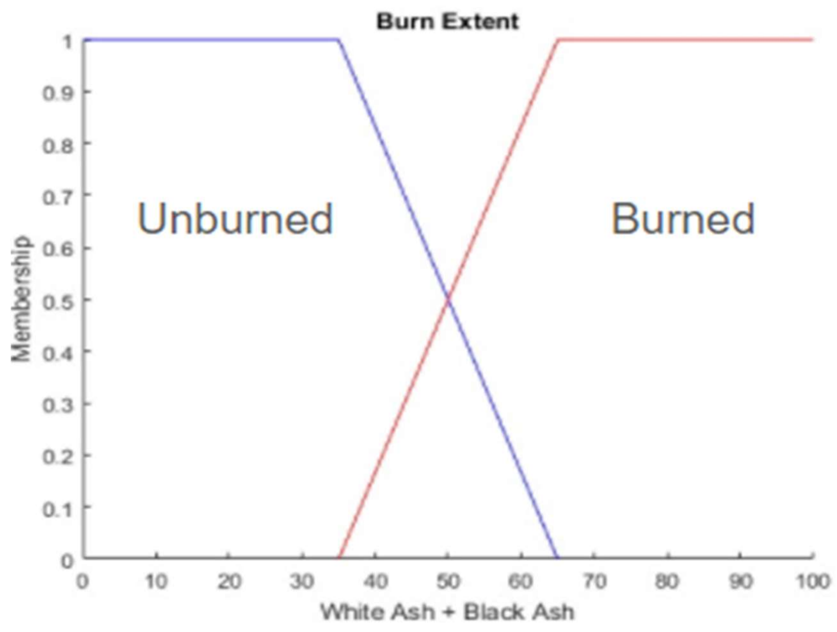


Figure 1 Fuzzy logic set membership for Burn Extent

As seen in Figure 1 burn extent, whether the ground is burned or not, has a set membership range from 35 to 65. So if 40% of the pixels are burned then there will be a .2 membership for burned and a .8 membership for unburned. It is also important to note that the set membership does not have to be set around 50%.

After the fuzzy logic was used to decreased the resolution, the accuracy of the classifier was measured for both the sUAS, using the original hyperspatial imagery, and the newly formed medium resolution image. The classification using hyperspatial imagery vastly outperformed the classification using sUAS imagery resampled to 30m by a margin of 93.16% to 56.62% for burn extent, burned compared to unburned. In terms of biomass consumption, low-intensity burn compared to high-intensity burn, hyperspatial classifications beat medium resolution classifications by 95.94% to 28.4% (Hamilton, Hamilton, & Myers, 2019).

2.2. sUAS compared to Landsat

Currently, there are several different ways to obtain imagery of fires. The two that pertain to our research are sUAS, more commonly known as drones, and satellites. The satellite that we will be focusing on is Landsat, a series owned and managed by NASA and the U.S. Geological Survey (USGS). It provides the longest continuous space-based imagery of Earth's lands. Information about Landsat and access to the free imagery, is available at landsat.gsfc.nasa.gov (Landsat Science, n.d.).

Both sUAS and Satellites have their advantages and disadvantages in various areas. The most significant difference is the spatial resolution. Spatial resolution is the size of the area on the ground which is captured within a single pixel in an image. The spatial resolution of images acquired with a DJI Phantom or a DJI Inspire, the two types of drones that NNU is currently using, is about 5cm when flying at 120 meters above ground level (AGL)(Hamilton, Bowerman, Colwell, Donahoe, & Myers, 2017). Both sUASs come with a digital color camera with a 94-degree view. It takes twelve-megapixel images with 3000 rows of pixels and 4000 columns of pixels. That compares to Landsat's spatial resolution of 30 meters.

Another difference between sUAS and Landsat is the availability of the imagery. With an sUAS, the availability is endless as long as the people flying follow the regulations put forth by the Federal Aviation Administration (FAA). These rules include but are not limited to, having someone certified to fly the sUAS and always having someone who is watching the sUAS. If the rules are followed then all that needs to be done is take the sUAS to where the research site.

Landsat, on the other hand, is more challenging to obtain imagery for, even though it is available at no cost. One of the places that can be used to download Landsat imagery is LandsatLook Viewer (LandsatLook Viewer, n.d.). It does a complete pass over the entire the globe every sixteen days. This can cause several problems. The first is that if the data that is needed is time sensitive and by the time Landsat is over it, then it may be too late to get an accurate image. The second problem is that if there are clouds or smoke over the area that is mapped, the image will be useless because the ground will not be visible, and it will be another 16 days until Landsat is over in the same place again (Hamilton, 2018). Accurate temporal resolution is important to BAER teams since they have 21 days post-suppression to gather data and write a burn recovery plan (WO Staff Program - Burned Area Emergency Response BAER. n.d.).

The limitation that arises with an sUAS is that their battery life will only give them the ability to fly small fires. The largest fire that was flown by our research team was the mile marker 106 fire, about 325 hectares of open land. Flying the fire took two drones, six people, 10 to 15 batteries and the better part of a day whereas that same burn would be a small part of a single image from Landsat. To take the next step and move to fires of class F (> 400 ha) and G (> 5,000 ha) (Size Class of Fire, n.d.), imagery from Landsat needs to be classified.

3. Methods

The goal of the project is to determine if the technologies used to classify burn severity from hyperspatial (5cm) can be used to classify burn severity from Landsat imagery. To accomplish this objective, hyperspatial orthomosaics needs a border of no value or null pixels so that the extent of the resampled sUAS imagery line up with the pixels of the

Landsat image. The pixels need to line up so that the training data created for classifications on Landsat imagery will match the pixels on Landsat.

Figure 2, Figure 3, and Figure 4 are a representation of a hyperspatial image overlaid over a Landsat image that illustrates the process of the project. Figure 2 illustrates the first problem that needs to be solved. That is that the projection for the Landsat image (orange) and the sUAS image (blue) are different. Projection is defined as how a model of a 3D globe gets mapped onto a 2D image. This makes it apear that the pixles are diaganal to each other.

Figure 3 illustrates the second problem, why the border of null (no data) is necessary. In the example the medium resolution image is orange and the hyperspatial image is blue. If there were no border when the hyperspatial image became resampled to 30 m, the pixels would not line up. There would be a gap between the resampled sUAS imagery and the Landsat imagery. The 30m resampled image, in this example, would be shifted a bit to the right and down from the Landsat image. Keep in mind that there are going to be far more pixels than in the example provided. When the border was created, there are in the range of several hundreds of pixels added to each side.

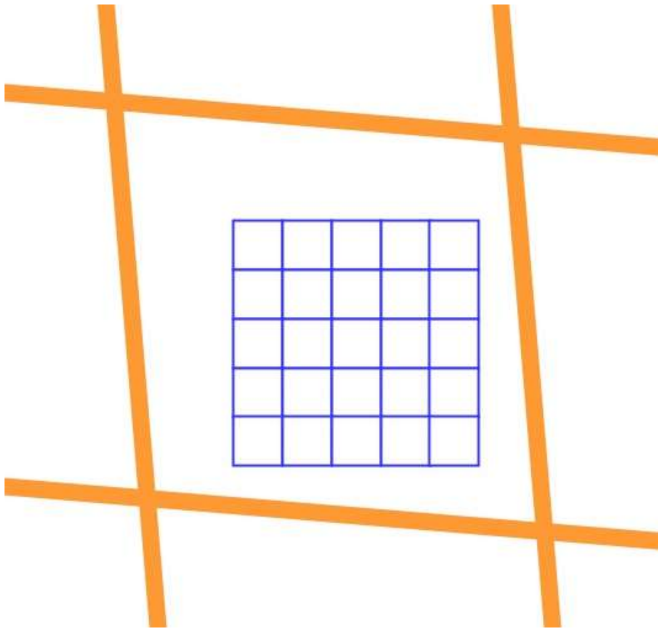


Figure 2 Image before projection is changed

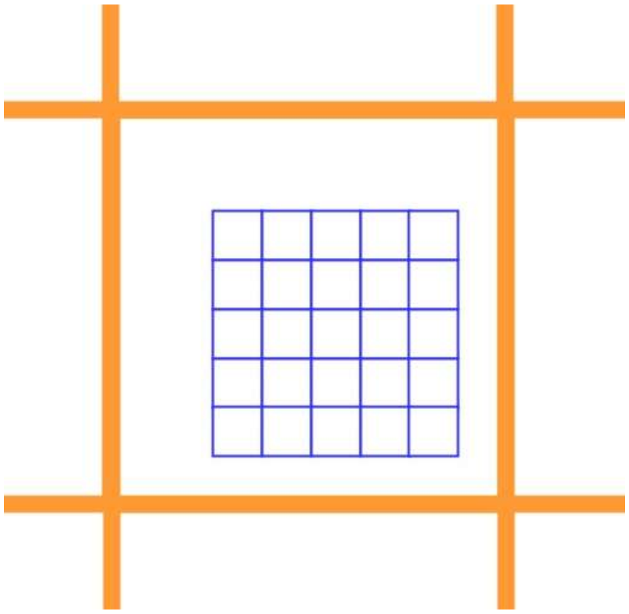


Figure 3 Image before the border is added

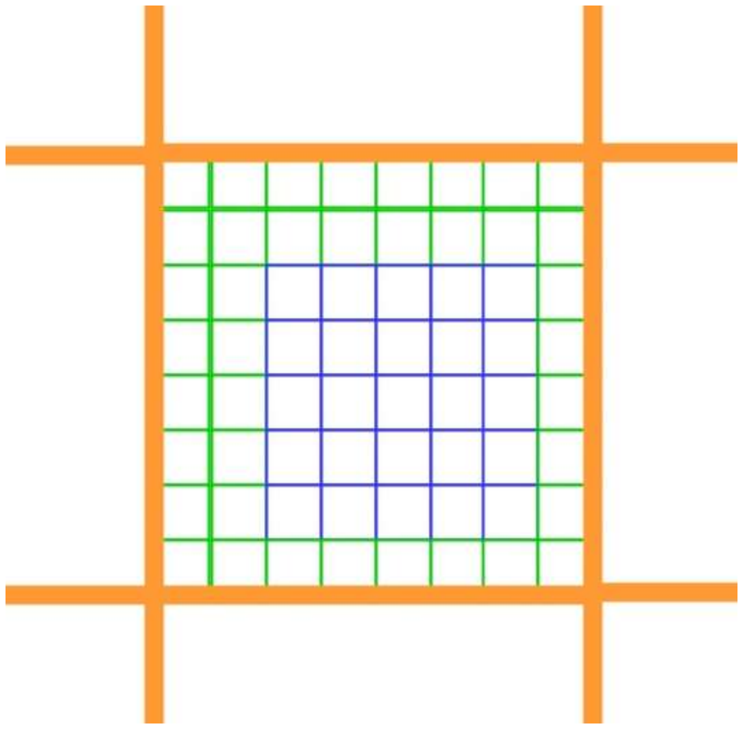


Figure 4 Image after the border is added

3.1. Finding the Size of the Border

There are two steps in making the border. The first step is to figure out how big the border needs to be. The calculation was accomplished using C++. Geospatial Data Abstraction Library (GDAL) was originally investigated to try to determine the size of the border. GDAL is an open source library for C, C++ and Python. GDAL is used to manipulate and get information from georeferenced images, images that have information telling where they are on the Earth (GDAL - Geospatial Data Abstraction Library, n.d.).

There were several reasons why in the end GDAL did not work for the project. The first reason was because GDAL did not contain the needed tools. Those tools being the ability to tell the location of the corners of the image. What was the final straw was that I discovered there was already a class called GeoImg that was created as part of this research effort at NNU, which with a few modifications could do what was needed.

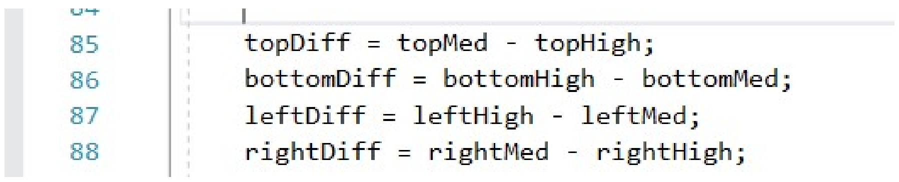
GeoImg was created in the Spatial Experiment to store and determine the georeferencing of an image (Hamilton, Hamilton, & Myers, 2019). In order for GeoImg to work, we needed to crop the Landsat image to the sUAS orthomosaic. GeoImg could already determine the coordinates of the top left of an image by reading the image's world file. The world file is one of the ways that an image knows its spatial reference. It contains the size of the pixels and the location of the centroid of the top left pixel. The calculation was already completed to determine the location of the actual top left by GeoImg (Hamilton, Hamilton, & Myers, 2019). From there it was a simple calculation (Equation 1) to determine the coordinates of the right and bottom of the image. The resolution and the number of rows were multiplied together and then added to the top to find the bottom. The same principal is used for finding the right of the image, but columns are used instead of rows and it is subtracted from the coordinate of the left of the image. The number of pixels was determined using the Open Source Computer Vision Library (OpenCV) library, a library that works with computer vision and machine learning. OpenCV is explain in more depth to follow in Section 4.2.

$$Bottom\ Location = \ Top\ Location - of\ rows * Spatial\ Resolution\ of\ rows$$

$$Right\ Location = \ Left\ Location + \#\ of\ collums * Spatial\ Resolution\ of\ collums$$

Equation 1 The formulas for finding the coordinates for the bottom and the right of the images

After calculating the location of the top, left, bottom and right of both the hyperspatial and medium resolution image, then a simple subtraction determines the size that the border needs to be. Since the cropped Landsat image surrounds the sUAS image on all sides, the coordinate of the hyperspatial are sometimes subtracted from the coordinate of the medium resolution and sometimes the reverse is true. Figure 5 The formulas for finding the size of the bordertells which happens for each side of the border.

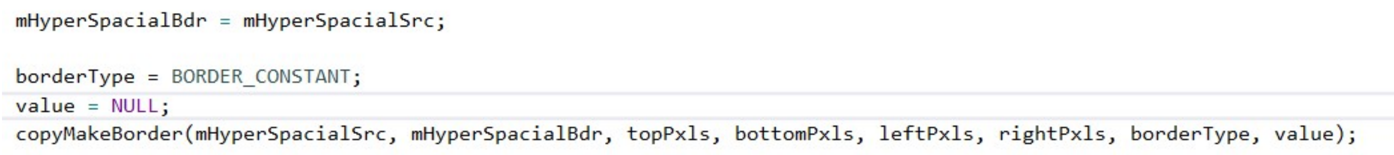


```
topDiff = topMed - topHigh;
bottomDiff = bottomHigh - bottomMed;
leftDiff = leftHigh - leftMed;
rightDiff = rightMed - rightHigh;
```

Figure 5 The formulas for finding the size of the border

3.2. Creating the Border

The border is made using OpenCV, an open source computer vision and machine learning library. OpenCV was utilized to create boarders for the images so that they were ready to classify. In this case, a function was called to create the border around the image. Figure 6 is the function call to copyMakeBorder() that was used to create the border.



```
mHyperSpacialBdr = mHyperSpacialSrc;

borderType = BORDER_CONSTANT;
value = NULL;
copyMakeBorder(mHyperSpacialSrc, mHyperSpacialBdr, topPxls, bottomPxls, leftPxls, rightPxls, borderType, value);
```

Figure 6 Making the border using OpenCV

The copyMakeBorder function parameter list and associated arguments consist of:

1. source image
2. destination image
3. length in pixles of border on the top
4. length in pixles of border on the bottom
5. length in pixles of border on the left
6. length in pixles of border on the right
7. border type

8. value [optional]

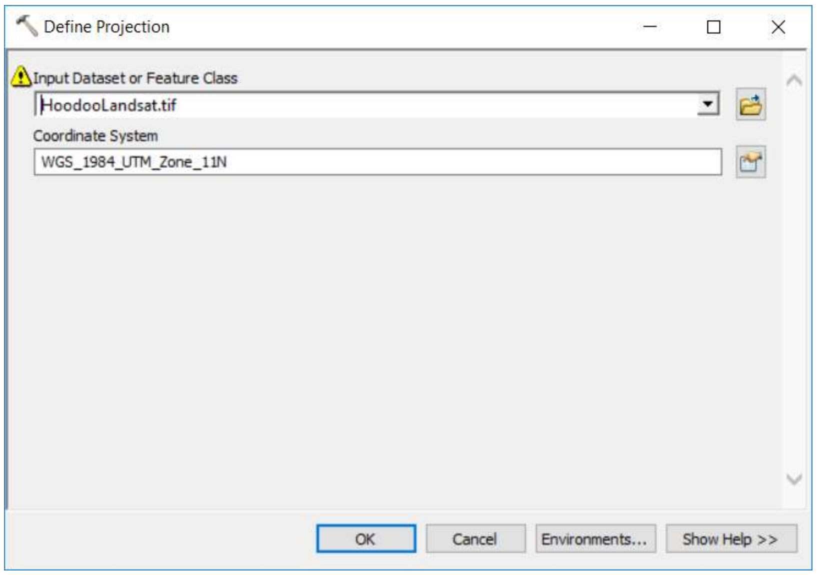
The first is the source image, mHyperSpacialSrc, and the second is the destination image, mHyperSpacialBdr. They both need to be the same size and type of image. The next four, topPxls, bottomPxls, leftPxls, rightPxls, are integers describing the size, in pixels, each side of the border is going to be. The second to last variable, borderType, determines the type of border created. The two options for borderType are BORDER_CONSTANT, the border to be one single value, or BORDER_REPLICATE, whatever the value of the outer pixel is in the image will be extended to the end of the border. If the program specifies BORDER_CONSTANT, another variable is needed to determine the value of the border. The final variable, value, fulfills that role (Adding borders to your images — OpenCV 2.4.13.7 documentation, n.d.). The program uses NULL to denote that the pixels in the border have no value. NULL, in the sense of red, green, blue color, has the value of 0,0,0. NULL is used as the value because when Pix4D, the program that is used to create orthomosaics from sUAS imagery, sets the pixles not in any images to NULL.

3.3. Modifying the Landsat Imagery

Several modifications need to happen to the Landsat imagery before it can be used to figure out the size of the border around the hyperspatial imagery. The first modification is that the projection of the medium resolution image needs to change to the same projection as the hyperspatial image. The second modification is to crop the Landsat image so that the extent of it is never more than one pixel past the end of the hyperspatial image.

Both reprojection and cropping can be done using the Define Projection tool (Figure 7) in ArcMap. Define Projection is located in the ArcMap toolbox under Data Management Tools, then Projections and Transformations. The dialog box will ask for the “Input Raster or

Feature Class" and the "Coordinate System." Chose the Landsat image for "Input Raster or Feature Class." Click the dialog box to the right of "Coordinate System" and open the Layers folder. Choose whichever cordinate system is used by the sUAS orthomosaic. Clicking on the plus to the left of the projection method will reveal which layers are using that projection.



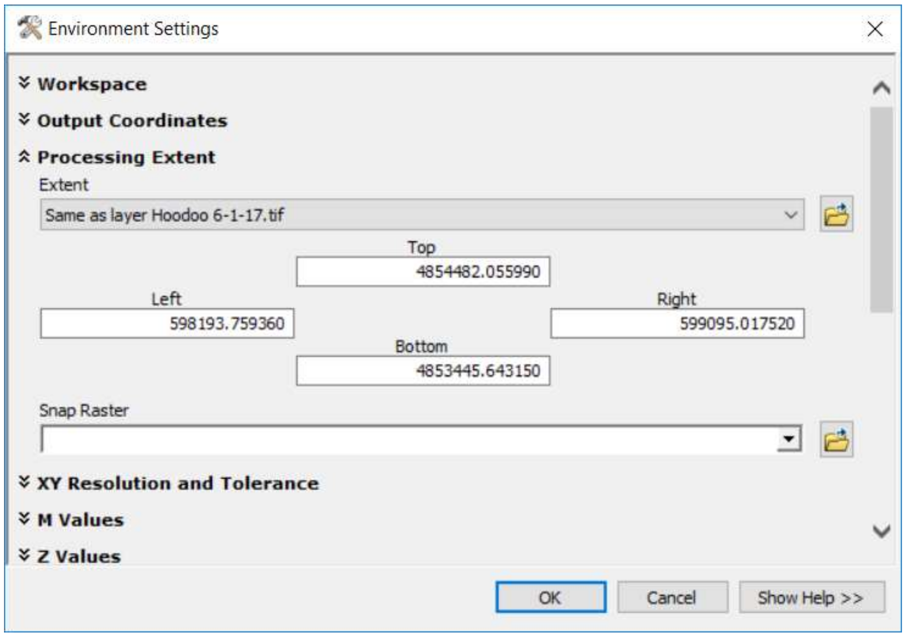
Define Projection

Input Dataset or Feature Class: HoodooLandsat.tif

Coordinate System: WGS_1984_UTM_Zone_11N

Figure 7 Define Projection Tool

This dialog also can crop the image. The crop tool is in the Environment Settings. In here go to Processing Extent (Figure 7). The Extent box specifies where the crop is going to happen. Selecting the hyperspatial layer will crop the Landsat image. The Landsat image will still be slightly larger than the sUAS image, but a new pixel will not start after the end of the sUAS image. The size difference is what the border is used to fix.



Extent: Same as layer Hoodoo 6-1-17.tif

Top: 4854482.055990

Left: 598193.759360

Right: 599095.017520

Bottom: 4853445.643150

Snap Raster:

Figure 8 Defining Processing Extent Tool

4. Results

The initial results obtained from the classification have been very promising. Results from the first fire worked perfectly and warrant future research. The border added to the sUAS orthomosaic lines up perfectly with the modified image obtained from Landsat. Figure 9 and Figure 10 illustrate this early success.

Figure 9 is the hyperspatial imagery from the sUAS. At this point, it does not have a border around it. The very pixilated image behind the sUAS image is Landsat which has already been modified so that its projection is the same as the sUAS and so that its extent is about a pixel greater than the extent of the sUAS image. The pixels for Landsat have a spacial resolution of 30 meters, so there is still a big difference in extents of the two images.

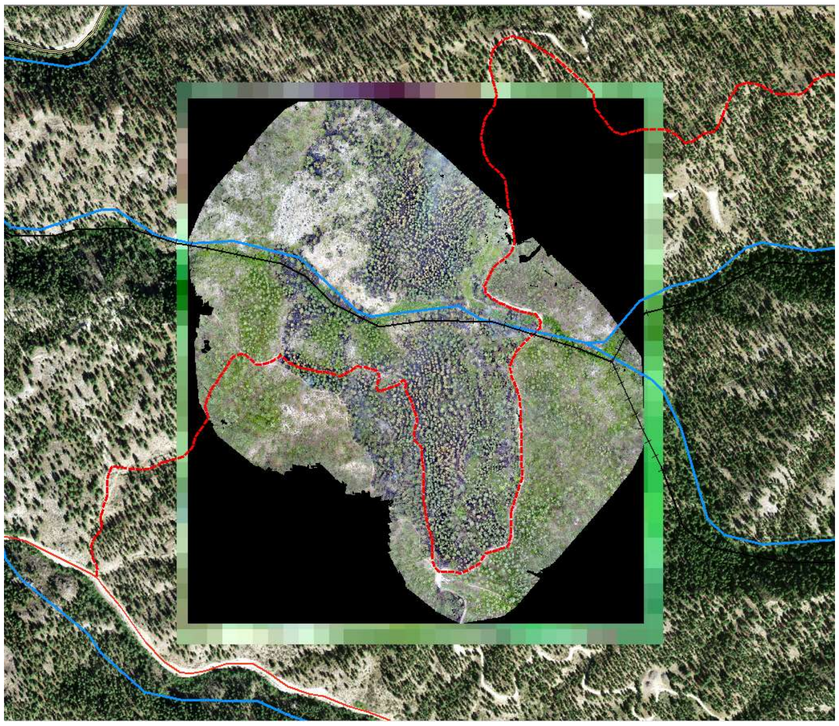


Figure 9 Hyperspatial orthomosaic without border overlaid over Landsat border

Some other aspects of the examples are the background and the lines across the images. The background is National Agriculture Imagery Program (NAIP) imagery. The USDA's Farm Service Agency (FSA) provides NAIP imagery with the goal of mapping the continental U.S. during the agriculture season every year if funding is available. FAS then makes the mosaic available to the government and the public about a year after the acquisition. NAIP imagery would not work for classifying imagery because it is renewed only once bi-annually (NAIP Imagery, n.d).

There are also several polyline geospatial map layers displayed over the images. They are there to show that the spatial references of the images match. The blue line represents a creek. The red line represents a road and the black line represents an old rail bed (Witherell, 1989).

In Figure 10 the border is added to the sUAS image. As you can see it perfectly lines up with Landsat image. From here the image can be reimaged to a lower resolution and used as training data for class G fires by using the methodology laid out in the Spatial Experiment (Hamilton, Hamilton, & Myers, 2019).

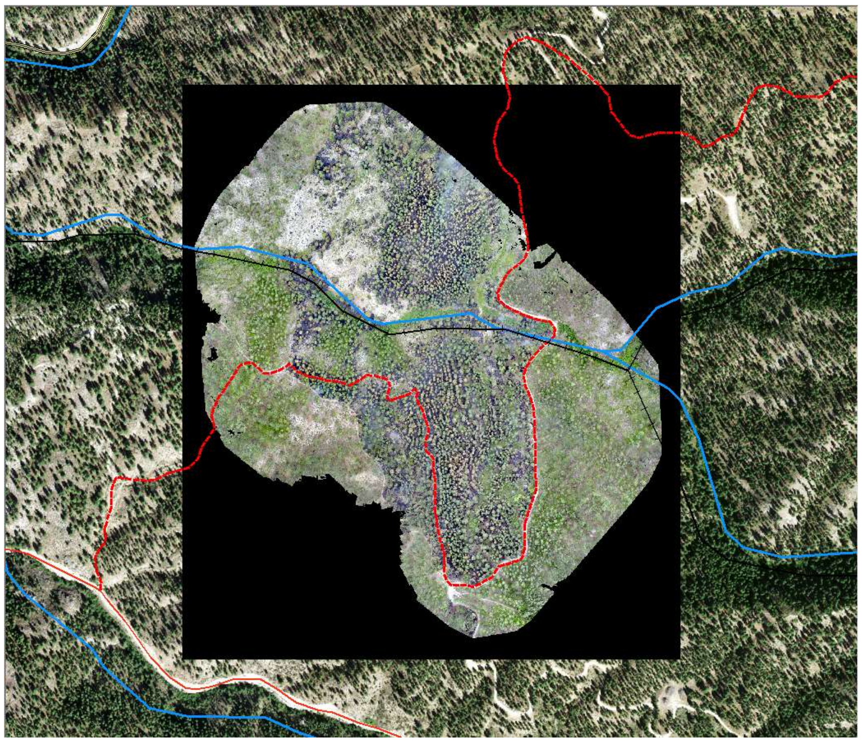


Figure 10 Hyperspatial orthomosaic with border overlaid over Landsat border

5. Conclusion

Overall the project was a success. GeoImg, with some modifications, determined the size of the border and OpenCV was able to create border. The border lines up with the clipped Landsat image. It is now ready to be applied to the methodology created in the Spatial Experiment.

I am happy that I had the opportunity to work on this project. It was challenging and frustrating at times. At other times it was very rewarding. I enjoyed getting to learn new soft skills and hard skills. I learned a lot about image manipulation and ArcGIS. I also learned how to work with teammates. Both of these will be very beneficial to me when I enter the workforce.

6. Future Work

There are several places that this project can go from here. The obvious one is data validation, which would include classifications of more fires. Merely having an image with a border around it does not help anyone. Where this becomes something that can be useful is when the classifications happens to it. These classifications would allow FireMAP to move away from only being able to map small fires to map Class F and G wildland fires.

Fire ecology is not the only discipline that could benefit from this tool. Both biomedical research into detecting cancer and the mapping of archaeological sights have benefited from FireMAP. Perhaps future research could investigate how this new tool could strengthen these fields.

References

Adding borders to your images — OpenCV 2.4.13.7 documentation. (n.d.). Retrieved April 10, 2019, from https://docs.opencv.org/2.4.13.7/doc/tutorials/imgproc/imgtrans/copyMakeBorder/copyMakeBorder.html

GDAL: GDAL - Geospatial Data Abstraction Library. (n.d.). Retrieved April 12, 2019, from https://www.gdal.org/index.html

Hamilton, D., Bowerman, M., Colwell, J., Donohoe, G., & Myers, B. (2017). A Spectroscopic Analysis for Mapping Wildland Fire Effects from Remotely Sensed Imagery. Journal of Unmanned Vehicle Systems, juvs-2016-0019. https://doi.org/10.1139/juvs-2016-0019

Hamilton, D., Hamilton, N., & Myers, B. (2019). Evaluation of Image Spatial Resolution for Machine Learning Mapping of Wildland Fire Effects. In K. Arai, S. Kapoor, & R. Bhatia (Eds.), Intelligent Systems and Applications (Vol. 868, pp. 400–415). https://doi.org/10.1007/978-3-030-01054-6_29

Hamilton et al. - 2017 - A Spectroscopic Analysis for Mapping Wildland Fire.pdf. (n.d.).

Hamilton et al. - 2019 - Evaluation of Image Spatial Resolution for Machine.pdf. (n.d.).

Heckendorn - Improving Mapping Accuracy of Wildland Fire Effect.pdf. (n.d.).

Heckendorn, R. (n.d.). Improving Mapping Accuracy of Wildland Fire Effects from Hyperspatial Imagery Using Machine Learning. 137.

Landsat Science. (n.d.). Retrieved April 9, 2019, from https://landsat.gsfc.nasa.gov/

LandsatLook Viewer. (n.d.). Retrieved April 10, 2019, from https://landsatlook.usgs.gov/

NAIP Imagery [Page]. (n.d.). Retrieved April 9, 2019, from temp_FSA_02_Landing_InteriorPages website: https://www.fsa.usda.gov/programs-and-services/aerial-photography/imagery-programs/naip-imagery/

OpenCV library. (n.d.). Retrieved April 10, 2019, from https://opencv.org/

Size Class of Fire | NWCG. (n.d.). Retrieved April 10, 2019, from https://www.nwcg.gov/term/glossary/size-class-of-fire

Unmanned Aircraft Systems (UAS). (n.d.). Retrieved April 10, 2019, from https://www.faa.gov/uas/

Witherell, J. (1989). The log trains of southern Idaho. Denver, Colo: Sundance Books.

WO Staff Program - Burned Area Emergency Response BAER. (n.d.). Retrieved April 10, 2019, from https://www.fs.fed.us/naturalresources/watershed/burnedareas.shtml

Appendix A: Code

1. Snapper.cpp

```
//opencv includes
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include "opencv2/imgproc/imgproc.hpp"

//c++ includes
#include <stdlib.h>
#include <stdio.h>

#include "GeoImg.h"
#include "Params.h"

#include <iostream>

using namespace cv;
using namespace std;

int main(int argc, char** argv) {
	Mat mHyperSpacialBdr;
	Mat mHyperSpacialSrc;
	Mat mHyperSpacialDisplay;
	Mat mMediumResoulution;

	GeoImg gMediumResolutionInfo;
	GeoImg gHyperSpacialInfo;

	gHyperSpacialInfo.setPath("C:\\Work\\Spacial\\Imagery\\Hoodoo\\HoodooOrtho\\Hoo
doo 6-1-17.tif", true);
	gMediumResolutionInfo.setPath("C:\\Work\\Spacial\\Imagery\\Hoodoo\\HoodoLandSat
_Reproj2.tif", true);

	gHyperSpacialInfo.writeWorldFile();
	gMediumResolutionInfo.writeWorldFile();

	double topHigh, bottomHigh, leftHigh, rightHigh;
	double topMed, bottomMed, leftMed, rightMed;
	double topDiff, bottomDiff, leftDiff, rightDiff;

	double topPxls, bottomPxls, leftPxls, rightPxls;


	int borderType = BORDER_CONSTANT;
	Scalar value;
	const char* window_name = "Snap drone to Landsat";
	RNG rng(12345);

	int listener;

	//load drone imagery
	mMediumResoulution = imread(argv[2], IMREAD_COLOR);// Read the file

	// Make sure image loaded
	if (!mMediumResoulution.data)
	{
		return -1;
		printf(" No data entered for hyperspacial , please enter the path to an 
image file \n");
	}


	//load LandSat imagery
	mHyperSpacialSrc = imread(argv[1], IMREAD_COLOR);// Read the file

	// Make sure image loaded
	if (!mHyperSpacialSrc.data)
	{
		return -1;
		printf(" No data entered for LandSat , please enter the path to an image 
file \n");
	}


	// create window without border
	namedWindow(window_name, WINDOW_AUTOSIZE);
	//imshow(window_name, mHyperSpacialSrc);
	//waitKey(0); // Wait for a keystroke in the window

	// add borders
	topHigh = gHyperSpacialInfo.getTop();
	leftHigh = gHyperSpacialInfo.getLeft();
	bottomHigh = gHyperSpacialInfo.getBotom(mHyperSpacialSrc.rows);
	rightHigh = gHyperSpacialInfo.getRight(mHyperSpacialSrc.cols);

	topMed = gMediumResolutionInfo.getTop();
	leftMed = gMediumResolutionInfo.getLeft();
	bottomMed = gMediumResolutionInfo.getBotom(mMediumResoulution.rows);
	rightMed = gMediumResolutionInfo.getRight(mMediumResoulution.cols);

	topDiff = topMed - topHigh;
	bottomDiff = bottomHigh - bottomMed;
	leftDiff = leftHigh - leftMed;
	rightDiff = rightMed - rightHigh;

	double resX = gHyperSpacialInfo.getResX();
	double resY = gHyperSpacialInfo.getResY();

	topPxls = topDiff / resY;
	bottomPxls = bottomDiff / resY;
	leftPxls = leftDiff / resX;
	rightPxls = rightDiff / resX;


	mHyperSpacialBdr = mHyperSpacialSrc;

	borderType = BORDER_CONSTANT;
	value = NULL;
	copyMakeBorder(mHyperSpacialSrc, mHyperSpacialBdr, topPxls, bottomPxls, 
leftPxls, rightPxls, borderType, value);

	//save image
	try {
```

```
              imwrite("C:\\Work\\Spacial\\Imagery\\Hoodoo\\OrthoBrdr.tif",
mHyperSpacialBdr);

       }
       catch (runtime_error& ex) {
              fprintf(stderr, "Exception converting image to TIFF format: %s\n",
ex.what());
              return 1;
       }

       //create Spatial Reference for Boarder image
       GeoImg gBoarderInfo;
       gBoarderInfo.setPath("C:\\Work\\Spacial\\Imagery\\Hoodoo\\OrthoBrdr.tif",
false);

       gBoarderInfo.setTopLeftX(gMediumResolutionInfo.getTopLeftX());
       gBoarderInfo.setTopLeftY(gMediumResolutionInfo.getTopLeftY());
       gBoarderInfo.setResX(gHyperSpacialInfo.getResX());
       gBoarderInfo.setResY(gHyperSpacialInfo.getResY());

       gBoarderInfo.writeWorldFile();

       //display image
       mHyperSpacialDisplay = mHyperSpacialBdr;
       resize(mHyperSpacialDisplay, mHyperSpacialDisplay,
Size(mHyperSpacialDisplay.cols / 32, mHyperSpacialDisplay.rows / 32)); // to half size
or even smaller

       imshow(window_name, mHyperSpacialDisplay);
       //imshow(window_name, mHyperSpacial);


       listener = waitKey(0);
       return 0;
}
```

2. GeoImg.h

```
#pragma once
#include <string>

using namespace std;

class GeoImg
{
private:
       string path;
       string baseName;
       string ext;
       double resX;
       double resY;
       //actual top left
       double topLeftX;
       double topLeftY;
public:
	GeoImg();
	~GeoImg();
	void setPath(string path, bool setSR);
	string getFilePath();
	string getFolder();
	string getBase();
	string getExt();


	void cloneResTL(GeoImg srcImg);
	double getTopLeftX();
	double getTopLeftY();
	double getResX();
	double getResY();

	void setTopLeftX(double loc);
	void setTopLeftY(double loc);
	void setResX(double res);
	void setResY(double res);

	int writeWorldFile();
	string getPrjFilePath();

	double getTop();
	double getLeft();
	double getBotom(int rows);
	double getRight(int cols);

private:
	string getWorldFilePath();

};
```

3. GeoImg.cpp

```
#include <iostream>
#include <fstream>
#include <string>

#include "GeoImg.h"

GeoImg::GeoImg()
{
}

GeoImg::~GeoImg()
{
}


void GeoImg::setPath(string path, bool setSR)
{
	//parse out path variable
	this->path = path.substr(0, path.find_last_of("\\"));
	if (path.find_last_of(".") != string::npos)
```

```
		{
			ext = path.substr(path.find_last_of(".") + 1);
			baseName = path.substr(path.find_last_of("\\") + 1,
				path.find_last_of(".") - path.find_last_of("\\") - 1);

		}
		else
			baseName = path.substr(path.find_last_of("\\")+1 ,
					path.find_last_of(".")- path.find_last_of("\\")-1);

		//populate member variables
		if (setSR)
		{
			string worldExt;
			ifstream file;
			if (ext == "tif")
				worldExt = "tfw";
			string fname = this->path + "\\" + baseName + "." + worldExt;

			file.open(this->path + "\\" + baseName + "." + worldExt);
			if (file)
			{
				string num;
				file >> num;
				resX = stod(num);
				file >> num;
				file >> num;
				file >> num;
				resY = stod(num)*-1.0;
				file >> num;
				topLeftX = stod(num)-(resX/2.0);
				file >> num;
				topLeftY = stod(num)+(resY/2.0);
				file.close();

			}
			else
			{
				cout << "ERROR: unable to open world file." << endl;
			}
		}
		int i = 0;

	}

	string GeoImg::getFilePath()
	{
		return this->path + "\\" + this->baseName + "." + this->ext;
	}

	string GeoImg::getFolder()
	{
		return this->path;
	}

	string GeoImg::getBase()
	{
		return this->baseName;
```

```
}

string GeoImg::getExt()
{
        return this->ext;
}

string GeoImg::getPrjFilePath()
{
        return this->path + "\\" + this->baseName + ".prj";
}

double GeoImg::getTop()
{
        return topLeftY;
}

double GeoImg::getLeft()
{
        return topLeftX;
}

double GeoImg::getBotom(int rows)
{
        double top = getTop();
        double botom = top - rows * this->resY;

        return botom;
}

double GeoImg::getRight(int cols)
{
        double left = getLeft();
        double right = left + cols * this->resX;

        return right;
}

//double GeoImg::getRight(int cols)
//{
//      double left = getLeft();
//      double right = left - cols * getResX();
//
//      return right;
//}

string GeoImg::getWorldFilePath()
{
        string worldExt;
        if (ext == "tif")
                worldExt = "tfw";
        else if (ext.empty())
                worldExt = "tfw";
        else
                cout << "ERROR: image format not tif" << endl;

        return this->path + "\\" + this->baseName + "." + worldExt;
```

```
}

double GeoImg::getTopLeftX()
{
	return this->topLeftX;
}

double GeoImg::getTopLeftY()
{
	return this->topLeftY;
}

double GeoImg::getResX()
{
	return this->resX;
}

double GeoImg::getResY()
{
	return this->resY;
}

int GeoImg::writeWorldFile()
{
	// open world file
	ofstream worldFile;
	string tfwPath = this->getWorldFilePath();

	worldFile.open(this->getWorldFilePath());
	// write settings
	worldFile << to_string(this->getResX()) << endl;
	worldFile << "0" << endl<< "0"<< endl;
	worldFile << to_string(this->getResY()*-1.0) << endl;


	double tlX = topLeftX - (resX / 2.0);
	double tlY = topLeftY - (resY / 2.0);
	worldFile << to_string(topLeftX + (resX / 2.0)) << endl;
	worldFile << to_string(topLeftY - (resY / 2.0)) << endl;
	worldFile.close();

	return 1;

}

void GeoImg::cloneResTL(GeoImg srcImg)
{
	this->resX = srcImg.resX;
	this->resY = srcImg.resY;
	this->topLeftX = srcImg.topLeftX;
	this->topLeftY = srcImg.topLeftY;
}

void GeoImg::setTopLeftX(double loc)
{
	this->topLeftX = loc;
}
```

```
void GeoImg::setTopLeftY(double loc)
{
	this->topLeftY = loc;
}

void GeoImg::setResX(double res)
{
	this->resX = res;
}

void GeoImg::setResY(double res)
{
	this->resY = res;
}
```

4. Params.h

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

using namespace std;

class Params
{
private:
	GeoImg srcImg;
	GeoImg destImg;
	int maxClass;
	int noDataClass;
	int resolution;
public:
	Params();
	~Params();

	void setSrcImg(string path);
	string getSrcImgPath();
	void setDestImg(string path);
	string getDestImgPath();
	string getDestImgFolder();
	string getDestImgBase();
	string getDestImgExt();


	void setRes(double res);
	double getRes();

	void setMaxClass(int maxClass);
	int getMaxClass();

	void setNoDataClass(int maxClass);
	int getNoDataClass();

	void setDestWorld(int fromSrc = 1, int writeWorldFile = 1);

	double getSrcResX();
	double getSrcResY();
};
```

5. Params.cpp

```
#include <fstream>

#include "Params.h"


Params::Params()
{
}


Params::~Params()
{
}

void Params::setRes(double res)
{
	this->resolution = res;
}

double Params::getRes()
{
	return this->resolution;
}

void Params::setMaxClass(int maxClass)
{
	this->maxClass = maxClass;
}

int Params::getMaxClass()
{
	return this->maxClass;
}

void Params::setNoDataClass(int noDataClass)
{
	this->noDataClass = noDataClass;
}

int Params::getNoDataClass()
{
	return this->noDataClass;
}

void Params::setSrcImg(string path)
{
	srcImg.setPath(path, true);
}

void Params::setDestImg(string path)
{
	destImg.setPath(path, false);
}
```

```
string Params::getSrcImgPath()
{
	return this->srcImg.getFilePath();
}

string Params::getDestImgPath()
{
	return this->destImg.getFilePath();
}

string Params::getDestImgFolder()
{
	return this->destImg.getFolder();
}

string Params::getDestImgBase()
{
	return this->destImg.getBase();
}

string Params::getDestImgExt()
{
	return this->destImg.getExt();
}

double Params::getSrcResX()
{
	return this->srcImg.getResX();
}

double Params::getSrcResY()
{
	return this->srcImg.getResY();
}

void Params::setDestWorld(int fromSrc, int writeWorldFile)
{
	if (fromSrc)
		destImg.cloneResTL(srcImg);
	else
		; // Get Top Left coordinates from another image

	destImg.setResX(resolution);
	destImg.setResY(resolution);

	if (writeWorldFile)
	{
		destImg.writeWorldFile();

		// copy prj file if it exists.
		std::ifstream    srcPrjFile(srcImg.getPrjFilePath());
		std::ofstream    destPrjFile(destImg.getPrjFilePath());

		destPrjFile << srcPrjFile.rdbuf();
		srcPrjFile.close();
		destPrjFile.close();
	}
}
```