NORTHWEST NAZARENE UNIVERSITY

Object Identification in Imagery Using Cluster Analysis

THESIS

Submitted to the Department of Mathematics and Computer Science

in partial fulfillment of the requirements

for the degree of

BACHELOR OF SCIENCE

Patrick James Richardson

2017

THESIS

Submitted to the Department of Mathematics and Computer Science

in partial fulfillment of the requirements

for the degree of

BACHELOR OF SCIENCE

Patrick James Richardson

2017

Object Identification in Imagery Using Cluster Analysis

Author: Patrick Richardson

Patrick Richardson

Approved: [illegible]

Dale Hamilton, Professor of Computer Science, Department of Mathematics and Computer Science, Faculty Advisor

Approved: [illegible]

Brice Allen, Engineering Lab Manager, Department of Physics and Engineering, Second Reader

Approved:

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

ABSTRACT

Object Identification in High Resolution Images.

RICHARDSON, PATRICK (Department of Mathematics & Computer Science),
HAMILTON, DALE (Department of Mathematics & Computer Science).

Mapping wildland fire behavior and its effects is becoming more responsive, affordable and safer than is currently possible with existing methods. The Fire Monitoring and Assessment Platform (FireMAP) project seeks to develop image acquisition methods, algorithms and software which will improve the mapping of wildland fires. FireMAP's goal is to quickly classify imagery and analyze wildland fire effects after it has been acquired. To accomplish this, FireMAP has developed programs and procedures that utilize big data, image processing, and machine learning techniques to extrapolate data efficiently out of high resolution imagery. This paper investigates the utility of grouping together pixels in imagery that are similar spectrally and spatially using clustering algorithms to form objects within the image. Utilizing these objects, it will be easier to make observations in higher resolution imagery and acquire data through machine learning to yield details about an object's size, shape, and texture for aiding the classification of that object. At this resolution, the imagery is data represented as objects that are comprised of pixels instead of being represented as pixels. FireMAP is researching whether this object based classification will improve classification accuracy and performance, reducing the time to classify.

ACKNOWLEDGEMENTS

I would like to first acknowledge Dale Hamilton for letting me join FireMAP to work on this project and advising me throughout its development. I would also like to thank Llewellyn Johnston for being an amazing coworker, getting me up to speed in FireMAP’s operations and helping me develop this project. I also want to thank FireMAP’s funding sources: Idaho Space Grant Consortium, and NASA Experimental Program to Stimulate Competitive Research (EPSCoR), as well as Dr. Greg Donohoe and Dr. Eva Strand from the University of Idaho, for your help with FireMAP’s development.

Table of Contents

Table of Figures

BACKGROUND

Mapping wildland fire behavior and its effects is becoming more responsive, affordable and safer than is currently possible with existing methods. The Fire Monitoring and Assessment Platform (FireMAP) seeks to develop image acquisition methods, algorithms, and software that improves the mapping of wildland fires. FireMAP's team is funded by and works in conjunction with, NASA, the University of Idaho, the Department of Interior, and the USDA Forest Service for the purpose of fire research.

FireMAP's goal is to quickly classify and analyze imagery after it has been acquired for use in examining fire effects. To accomplish this, FireMAP has developed programs and procedures that utilize big data, image processing, and machine learning techniques to extrapolate data efficiently out of high-resolution imagery. This research will enable the acquisition, storage, and analysis of hyperspatial multi-spectral UAS imagery, utilizing image processing and machine learning techniques to map wildland fire effects in a more responsive, affordable and safe manner than is possible with current methods.

This is a project through Northwest Nazarene University's FireMAP program in an effort to develop methods, analytic tools and metrics to improve the mapping of post fire effects using hyperspatial (sub-decimeter) imagery acquired with small unmanned aircraft systems (sUAS). The goal of this project is to take an image and group similar contiguous pixels in it together to form objects. With objects it is easier to extrapolate useful data and make observations in higher resolution imagery. This project is expected to accurately group pixels together that are similar spectrally and spatially. These objects will represent actual objects in the imagery thus improving the image classification process. This project breaks grounds in that imagery at this resolution is data represented as objects that are comprised of pixels instead of being represented as pixels. Utilizing machine learning, object details such as size, shape, and texture can be developed and utilized as inputs for classification and data analysis purposes.

FireMAP is in the process of developing the algorithms for image acquisition and analysis. This project, thus, involves algorithm research and designing processes to aid development, specifically working on core analytics using cluster analysis to improve the mapping of wildland fires in acquired imagery. Researching the background in addition to looking into the main mechanisms and approaches of cluster analysis is the first major step in pursuing object-based classification.

RESEARCH

Points of Interest

The research began by searching for and examining existing techniques pertaining to grouping data together. At the time no other papers or algorithms were found in the public domain that closely resembled the goals of this project. Initial points of interest included looking into vector quantization, color quantization, and exploring various types of clustering methods that would suit the project's needs.

Vector quantization is a classic quantization technique originating from signal processing that allows the modeling of probability density functions (PDF) by utilizing the distribution of preliminary data. This type of quantization works by dividing a large set of data points into groups having approximately the same number of data points closest to them, with each group being represented by its centroid point. The centroid is the average position of all the points in all coordinate directions, with at least the X & Y directions for imagery. This technique is especially powerful for identifying the density of a large number of data points as well as high-dimensioned data. Vector quantization is commonly used for data compression, data correction, pattern recognition, density estimation, and clustering, thus forming an important basis for most clustering models. This technique can be used to help further the project in grouping large amounts of data points with multi-spectral data.

Color quantization is a process used in computer graphics that reduces the number of distinct colors used in an image with the intention that the new image be as visually similar as possible to the original image. This technique allows images with many colors to be displayed on devices that can only display a limited number of colors. It also acts as an efficient compression method for certain types of images. Figures 1 and 2 provide an example of limiting colors in an image using color quantization. Common algorithms for color quantization are also based on various clustering methods. This is an important principle for this project since high-resolution imagery is being used and reducing the variation or size of data may be a necessity when grouping data points.

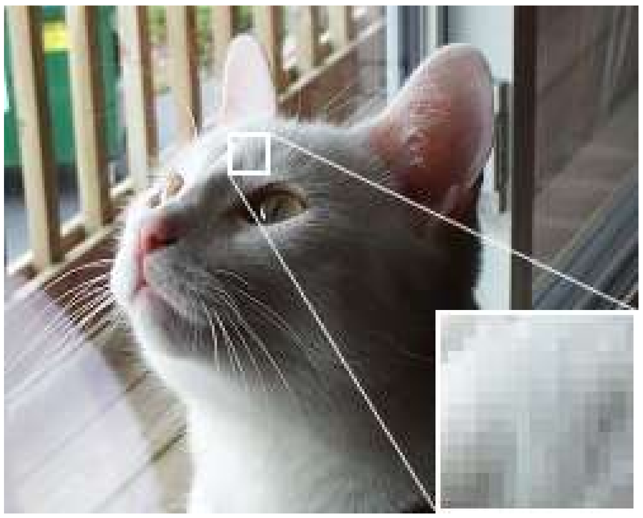


Figure 1. An example image in 24-bit RGB color

Taken from: Color quantization. (2017, January 11). Retrieved from

https://en.wikipedia.org/wiki/Color_quantization

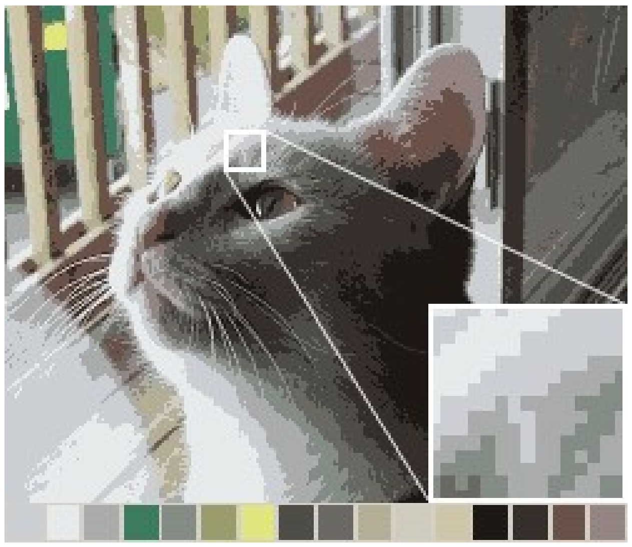


Figure 2. The same image reduced to a palette of 16 colors

Taken from: Color quantization. (2017, January 11). Retrieved from

https://en.wikipedia.org/wiki/Color_quantization

The third technique selected is cluster analysis which focuses on utilizing algorithms to group data. Cluster analysis, or unsupervised machine learning, is the task of grouping a set of objects in a way that puts similar objects into the same group, called a cluster. It is a main task of exploratory data mining and is a common technique for statistical data analysis. Cluster analysis is used in many fields, including machine learning, pattern recognition, image analysis, information retrieval, bioinformatics, and data compression. Cluster analysis is not one specific algorithm, but rather refers to the general clustering task that needs to be solved. Cluster analysis seeks to define what constitutes a “cluster” and how to efficiently find them, and so is a multi-objective optimization problem. Proper clustering algorithm and parameter settings depend on the individual data set and intended use of the results. Due to this, there are fundamentally different types of clustering, such as: hierarchical versus partitional clustering, overlapping versus fuzzy clustering, complete versus partial clustering, as well as density based clustering. It is an iterative process of knowledge discovery that involves trial and error. FireMAP will utilize the aforementioned techniques to enhance its capabilities throughout its various image analysis and classification procedures.

Clustering Methods

To properly determine which methods of clustering would best suit the project all of them need to be researched, where those that provide faster and more accurate results can be candidates for the design of the project. However, there is a wide variety of clustering algorithms so for initial research only the more basic techniques will be considered. First, the differences between hierarchical and partitional clustering will be considered to maintain an accurate clustering of objects. Hierarchical clustering repeats a cycle of either merging smaller clusters into larger ones or dividing larger clusters to smaller ones, this produces a hierarchy of clusters called a dendrogram (see Figure 3). The number of clustering groups depends on the hierarchal level chosen within the dendrogram.

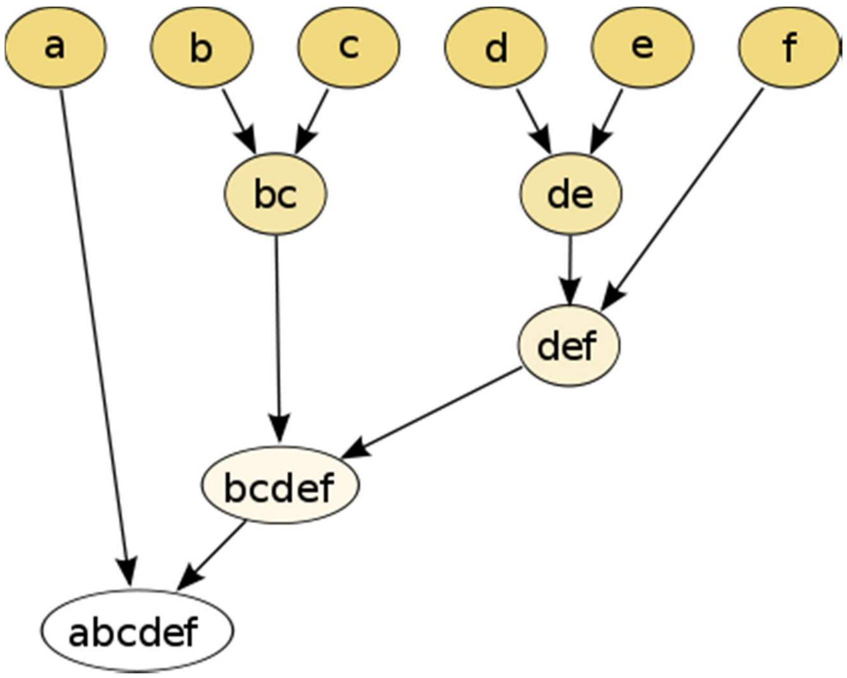


Figure 3. An example of a dendrogram

Taken from: Hierarchical clustering. (2017, May 05). Retrieved from https://en.wikipedia.org/wiki/Hierarchical_clustering

Partitional clustering generates various partitions and then evaluates them by some criterion. These are considered "nonhierarchical" since each data point is placed in exactly one mutually exclusive cluster. The main feature of this type of clustering is that it requires an input for the desired number of clusters to be generated, as opposed to having a dendrogram and needing only to select the proper level. One or both methods could be useful for finding objects in imagery when designing algorithms for this project.

Next, the differences between overlapping and fuzzy clustering must be considered to maintain an accurate clustering of data points so that objects can be represented appropriately. Overlapping is a non-exclusive clustering method that is used to reflect the fact that data can simultaneously belong to more than one group, where an exclusive clustering method assigns each data point to a single cluster. Fuzzy clustering is a form of exclusive clustering that assigns data to a cluster by using a membership weight that is between 0 (absolutely does not belong) and 1 (absolutely belongs). By examining this membership weight it is easier to determine which clustering group a data point belongs to, where determining this weight is based on some criterion. Both are valid methods that may have use in designing algorithms for this project.

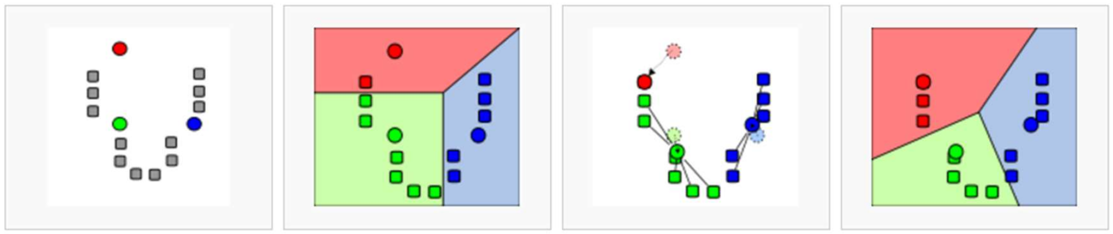
To further refine proper object acquisition in imagery, complete and partial clustering must be considered. These methods of clustering are fairly straightforward and yet they each define very different ways to treat data. Complete clustering is when every data point is assigned to a cluster, where in partial clustering this is not necessarily the case. Given certain scenarios, some data points could skew results in clustering procedures should they be kept, or some data points could have duplicate data that might slow down processing speeds. Partial clustering removes these points producing a better result. Each of these methods could prove useful for this project so both will be considered.

A final method in clustering is to consider whether or not to group data based on its density. Density-based clustering differs from centroid-based clustering in that it identifies "dense" clusters of data points, allowing easier discovery of clusters with arbitrary shapes and identifying outliers in the data. These outliers can then either be considered valuable data points or noise that needs to be ignored, which forms a basis for partial clustering. This technique does, however, usually requires more processing time and memory due to the more complex nature of this method. Given the high-resolution imagery and the arbitrary shapes of most objects, this project will most likely utilize this method.

Clustering Algorithms

Three primary types of algorithms: K-Means, Agglomerative Hierarchical, and Density-Based Spatial Clustering of Applications with Noise (DBSCAN) were found while researching clustering methods. Further research into these common clustering types was undertaken to assess their usefulness and to determine whether a more advanced type of clustering algorithm would be needed.

K-Means clustering is a method of vector quantization and partitional clustering that is popular for cluster analysis in data mining. This kind of clustering aims to partition data points into clusters in which each data point belongs to the cluster with the nearest mean, serving as a prototype of the cluster, represented by centroids (centroid model). K-Means clustering is a common algorithm known in cluster analysis as a “centroid model” where each cluster is a single mean vector. This method of clustering requires a user-specified number of clusters to be entered, then the algorithm will partition the data into the closest cluster group it matches. See Figure 4 for the general procedure of this clustering model. This type of clustering would work well given the project’s needs, however automating a proper clustering number for various high-resolution imagery could prove to be a challenge.



1. *k* initial "means" (in this case *k*=3) are randomly generated within the data domain (shown in color).

2. *k* clusters are created by associating every observation with the nearest mean. The partitions here represent the Voronoi diagram generated by the means.

3. The centroid of each of the *k* clusters becomes the new mean.

4. Steps 2 and 3 are repeated until convergence has been reached.

Figure 4. Demonstration of the standard K-Means algorithm

Taken from: K-means clustering. (2017, April 20). Retrieved from https://en.wikipedia.org/wiki/K-means_clustering

An agglomerative clustering strategy uses a "bottom-up" approach of merging smaller clusters into larger ones. Each data point starts as its own cluster, and pairs of clusters are merged together as one moves up the hierarchy of the dendrogram. This type of clustering is known as a "connectivity model" since it usually builds based on the distance connectivity between points. Ultimately unless a hierarchy is chosen all points will eventually become one big cluster. This type of clustering could also work well given the project's needs, however again automating a proper hierarchy level for various high-resolution imagery could prove to be a challenge.

Perhaps the most advanced of the basic clustering types would be the DBSCAN approach. This type of clustering is a "density model" in that it determines clusters based on connected dense regions in the data space. DBSCAN is particularly helpful in that it can determine the proper number of clusters through the algorithm, and it omits low-density data (noise). However, this can also a downfall since the data set overall could be fairly sparse or complete clustering could be required. Due to the more ambiguous nature of the high-resolution imagery, this type of clustering algorithm could prove to be the best given the project's needs and large data set. See Figure 5 for an example of DBSCAN clustering, where the solitary gray points are considered noise.

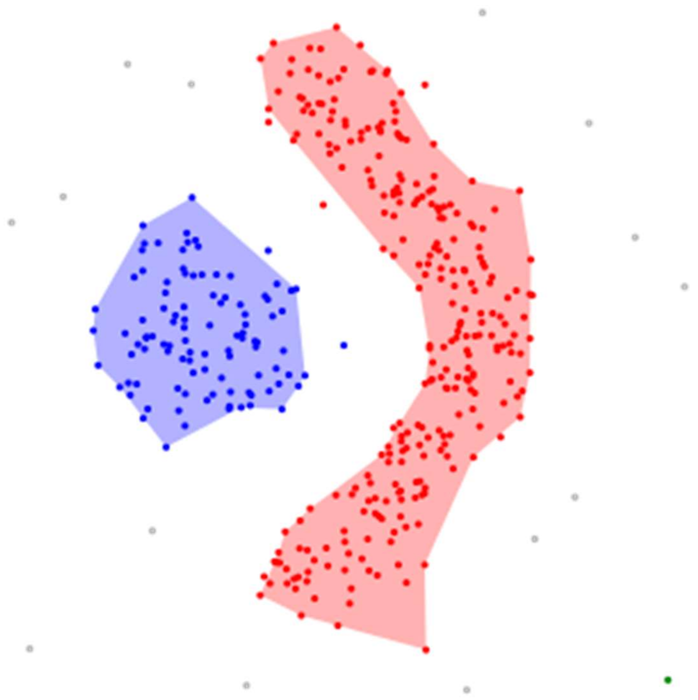


Figure 5. DBSCAN finding non-linearly separable clusters

Taken from: DBSCAN. (2017, May 04). Retrieved from https://en.wikipedia.org/wiki/DBSCAN

DESIGN & DEVELOPMENT

The various clustering techniques and algorithms found can now be used to develop efficient object based classification procedures for FireMAP. FireMAP is currently using OpenCV in its infrastructure for computer vision applications, due to its computational efficiency and a strong focus on real-time applications. This is an open source computer vision and machine learning software library with algorithms that can be used to detect and recognize faces, track camera movements, and more with over 2500 optimized algorithms already in place.

After some deliberation and research into OpenCV and the various clustering techniques it provides, it appears that the actual object identification process would require starting with the clustering of non-spatial dimensions within an image, allowing the examination of colors and other layers of multi-spectral data. This step would normalize the imagery before trying to do proper data clustering for objects and would be similar to normalization in image processing. Normalization here involves changing the range of pixel values so that the image is more familiar or normal, given the current needs of the project, to achieve consistency in the set of data.

Spectral Clustering

Though all of the above clustering types could provide excellent ways to obtain detailed information for this project, ultimately only one needs to be used for grouping the data spectrally at this time since more advanced algorithms and combinations can be used with more research and testing. OpenCV has only K-Means clustering built into it, whereas MATLAB and most other libraries have K-Means and Hierarchical based clustering methods. This might persuade the project to divert from OpenCV, however, it is already so ingrained in FireMAP that OpenCV's K-Means method will be tested first in an effort to help keep FireMAP elements compatible. In running K-Means through example images and programs it became clear that the algorithm could either split up the majority colors in an image or normalize the image based on those majorities. See Figures 6-8 for an example of splitting colors and normalizing the image using this algorithm. Separating the colors in an image is not particularly helpful to the project, although the normalization that can result looks rather promising for spectral clustering. There are a few issues with this approach in that noise can exist within resulting clusters or the algorithm itself can have a long runtime due to the number of resulting clusters required.

Using certain filters such as a blur in OpenCV could be a more reliable option for general normalization, however, some spectral details can be lost in this process as well. Overall, a K-Means algorithm works well at breaking up the data, but the user has to pick the right number of clusters initially to get proper results. It has become apparent through this testing that complete clustering is better suited for data analysis than partial clustering because the dataset is too important to risk removing any of it. It also became clear that fuzzy clustering is too much work for what it provides given too much overhead and computation time additions, however, an exclusive clustering method is still more useful than a non-exclusive one. Some additional testing into hierarchical and density-based clustering is necessary before deciding on any particular clustering type.

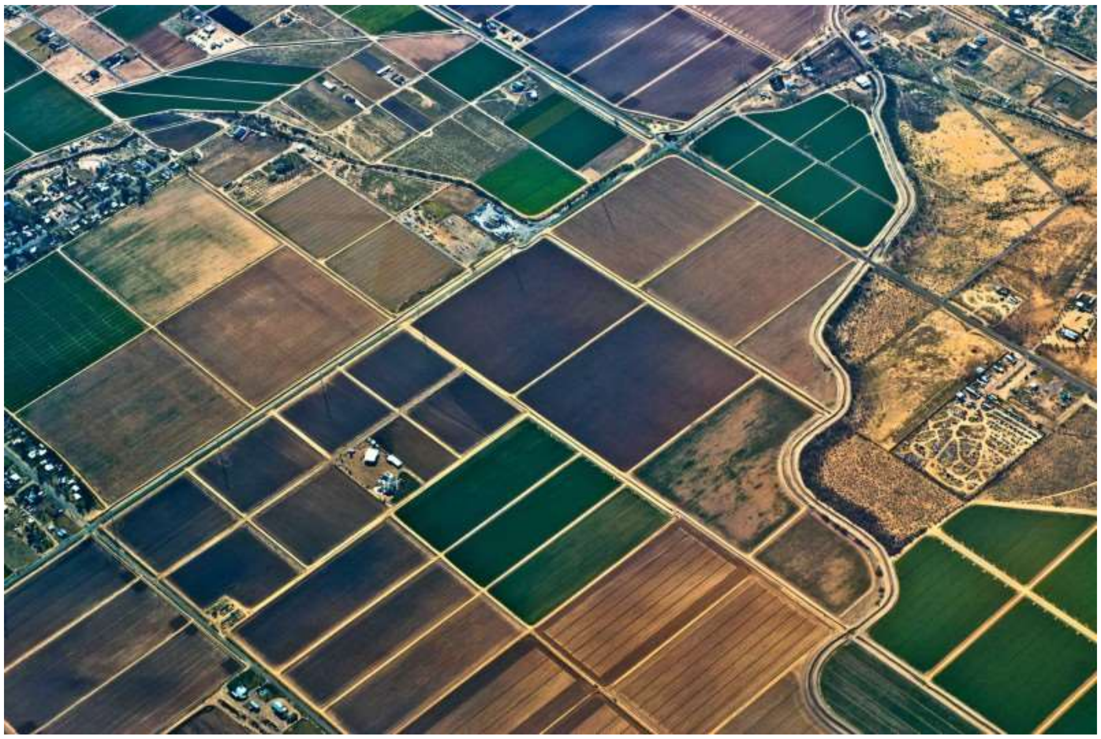


Figure 6. Smaller and simpler example image for testing K-Means.

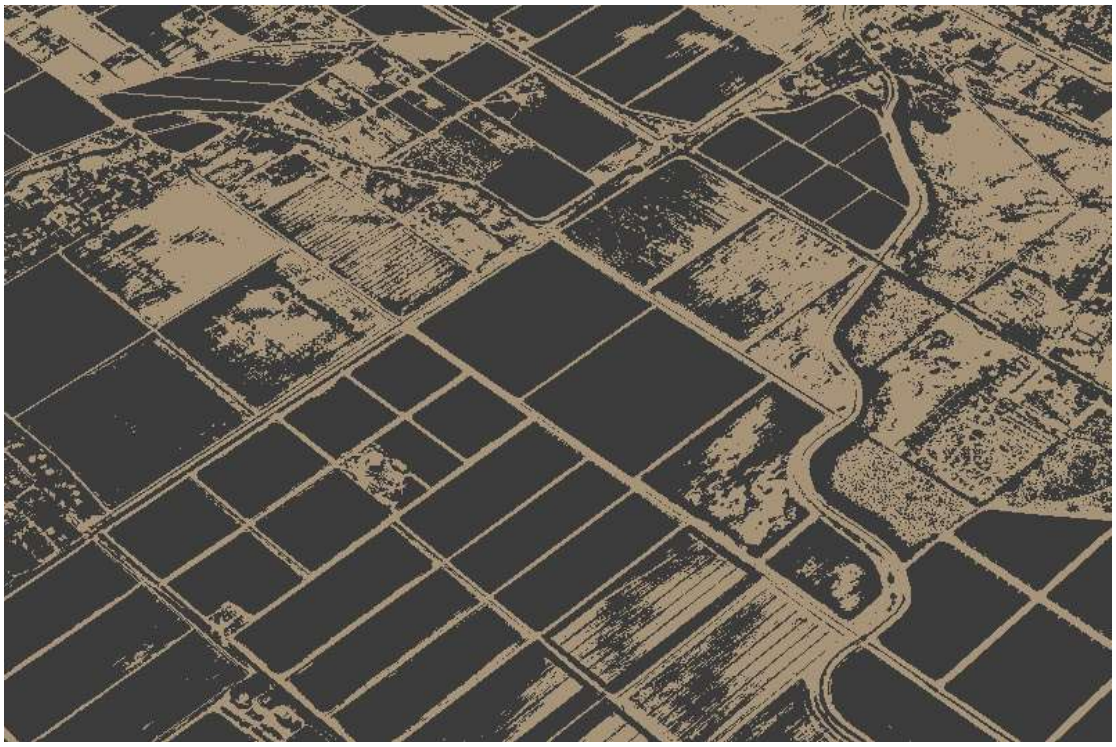


Figure 7. This is Figure 6 run through the K-Means algorithm to split major colors.

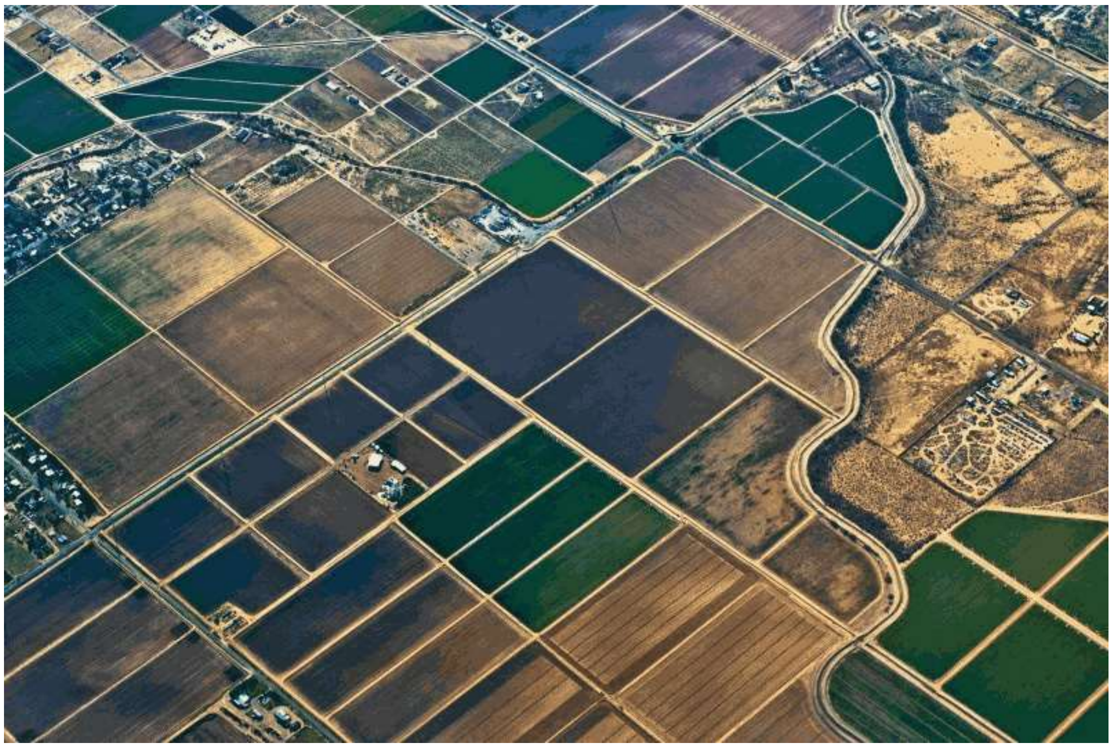


Figure 8. This is Figure 6 run through the K-Means algorithm to normalize the image, thus reducing variation in data for easier manipulation.

After looking into OpenCV’s K-Means algorithm and running s few tests with it, some questions came up about why reducing pixel variation through normalization seemed to be the best idea. It is easier to cluster data with less variation but details are lost through a normalization process. With this starting point, additional research went into exploring an easier way to map the variations as opposed to limiting them, specifically utilizing a variation of principal component analysis (PCA). PCA is a statistical procedure that uses a transformation to convert a set of data into a set of values that are linearly uncorrelated variables called principal components. What this means is that data with really high variations can be converted into a representation that shows each data point’s variation compared to every other data point’s variations. For example, in Figure 9 there are a bunch of data points that vary widely across this coordinate system, but using PCA it is possible to assign a new coordinate system (represented by the red arrows) to see the data variation in a more useful way.

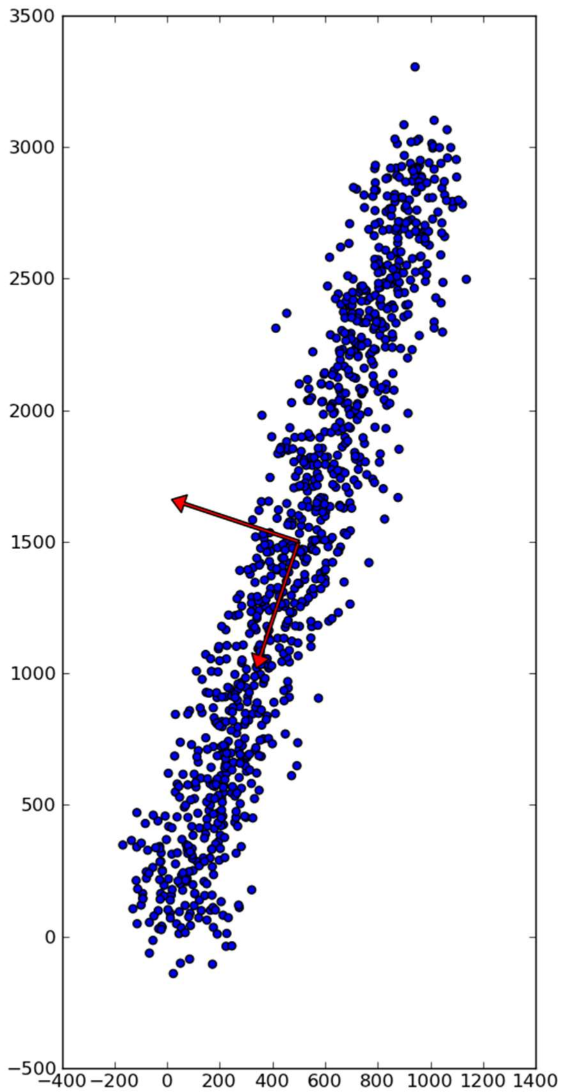


Figure 9. Example of PCA mapping data

Taken from: Basic example for PCA with matplotlib. (2016, April 19). Retrieved from http://stackoverflow.com/questions/18299523/basic-example-for-pca-with-matplotlib

A very useful and common form of PCA is a method that uses eigenvectors and eigenvalues. Eigenvectors are linear transformations of data that form a vector space, as shown in the previous PCA figure. However, eigenvalues yield a crucial element towards mapping spectral data that eigenvectors alone cannot. These are associated with each eigenvector and hold a value that represents how much the data is changed. Together this is a distribution model that performs its operations based on statistical distributions. This provides a new coordinate system (eigenspace) for the data that shows where the variation in the data as well as by how much it varies, providing both vector and color quantization. Figure 10 gives a visual representation of an altered image using eigenvectors and eigenvalues. Plotting points in the eigenspace allows for nicer manipulation with the data because it allows the data to be transformed into a model that is easier to see and work with compared to the original dataset.

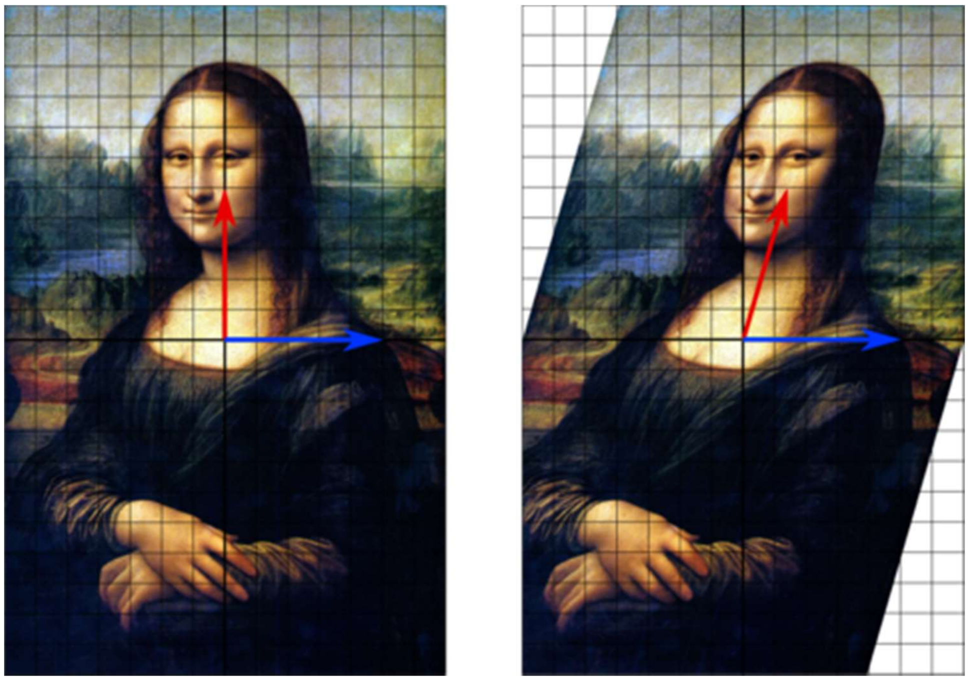


Figure 10. Example of shifting an image and the resulting eigenspace

Taken from: Eigenvalues and eigenvectors. (2017, May 03). Retrieved from https://en.wikipedia.org/wiki/Eigenvalues_and_eigenvectors

Some problems emerged while testing the principle of utilizing the eigenspace for exploring data variation. While OpenCV has existing algorithms to calculate eigenvectors and eigenvalues, the resulting affinity matrix of information was incredibly huge. Figures 11-12 show a test example of this matrix. This affinity matrix is a translator for converting between the eigenspace and the regular space of data. The resulting affinity matrix in Figure 12 represents each data point's eigenvector and eigenvalue obtained from Figure 11 in a matrix form that compares every data point's results to each other. Where Figure 11 at one-tenth the size is 625 data points and the resulting matrix of Figure 12 contains roughly 390,625 pieces of information related to that data. For this application, it is not feasible to use this procedure given that it requires an exponential amount of memory and time to produce the matrix based on the size of the imagery. However, this is incredibly accurate and useful but only once memory limitations are surpassed and processing speeds increased.

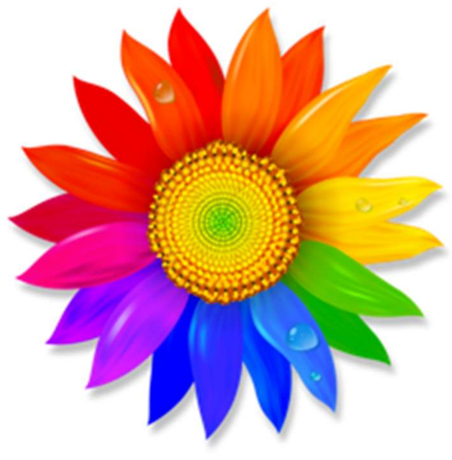


Figure 11. Very small, multi-colored image for testing variation

Taken from: Màu sắc kì diệu. (n.d.). Retrieved from http://simplecarry.com/blogs/blog/mau-sac-ki-dieu

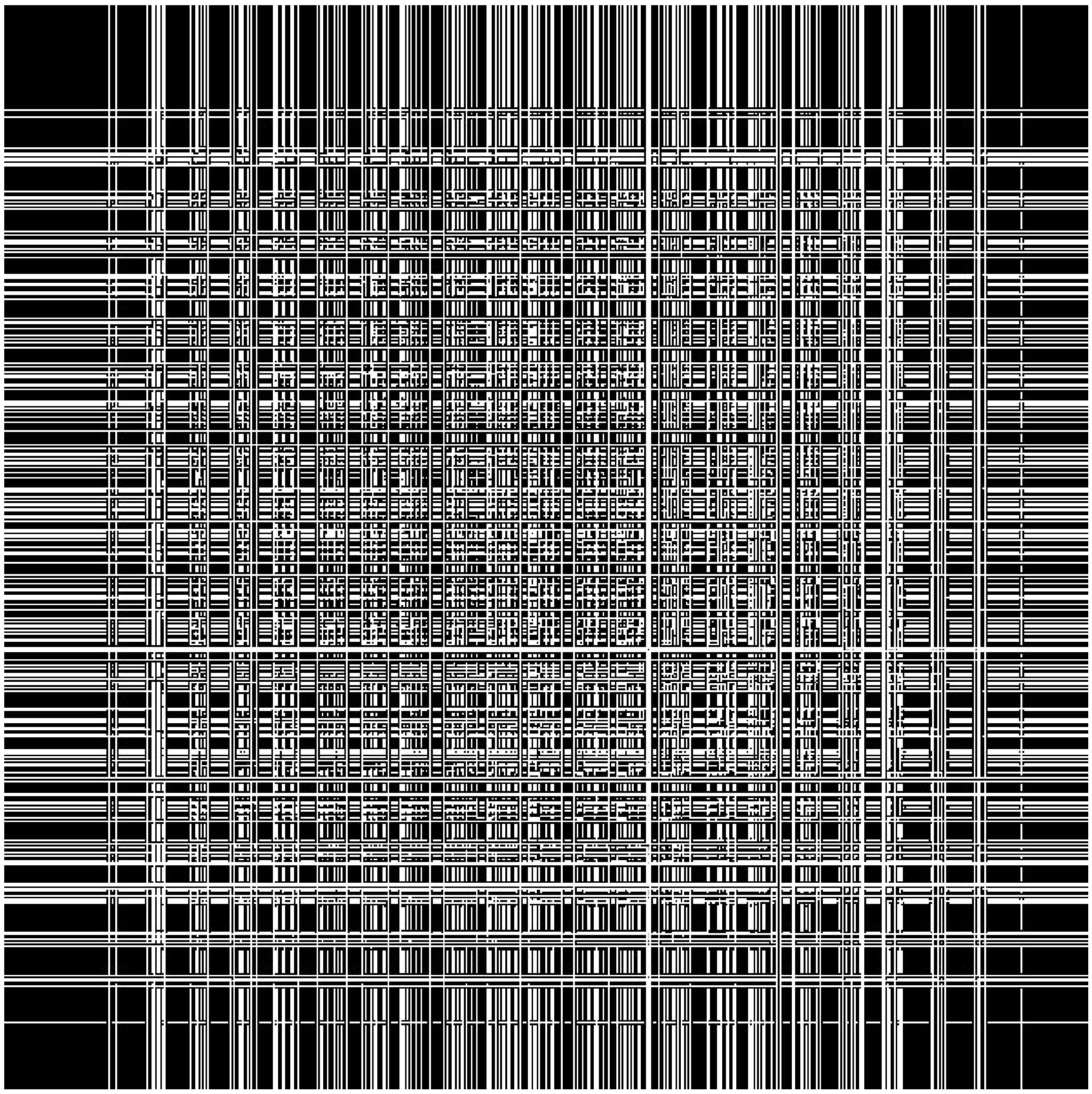


Figure 12. Resulting affinity matrix where Figure 11 is reduced to one-tenth the size.

Spatial Clustering

Once the clustering of non-spatial dimensions is complete, the FireMAP cluster analytics will then need to go another step further and segment the clusters based on spatial proximity to fully acquire distinct object based data. By using clustering methods to group pixels spectrally and spatially it should be possible to detect objects in an image.

Identifying these objects can yield the size, shape, type, and color, as well as additional object attributes depending on what is necessary, for machine learning classification. After additional research and application into spatial proximity, it will then possible to commence the designing of any more algorithms and processes for FireMAP to enhance its image acquisition and analysis needs. Care needs to be taken so that the system remains cohesive, organized, and functional.

After examining the various clustering types it appears that DBSCAN is best suited for spatially clustering the resulting spectral clusters. By using DBSCAN it is possible to find all the natural objects compared to choosing the number of groups or a hierarchical level. This algorithm will also be better at finding non-geometric shapes, as well as either omitting low-density data or absorbing it into nearby clusters depending on the project's needs. However, this algorithm will be slower for accuracy and clustering multiple attributes simultaneously may cause problems. Another concern is that this particular algorithm is more complex and is not integrated into most libraries, such as OpenCV. This will cause issues trying to adopt this method within OpenCV's environment and existing class structures due to difficulty in creating an algorithm within a non-standard library. Pursuing this type of clustering yields a lot of useful information regarding the handling of spatial clustering, however, this algorithm does not work as ideally as anticipated and may not work for this project's needs.

In trying to determine the best way to spatially cluster information, research towards new algorithms will be necessary. Connected components is one algorithm that may prove extremely helpful in finding objects spatially. This concept is used in graph theory to keep track of connected vertices and information pertaining to those connections. It starts searching at some particular vertex and finds all connected vertices in the group of data. Should any vertices not be found it will repeat the process on the remaining vertices. In this setting, it will be most helpful running this algorithm on each spectral group to find objects of similar color near each other. See Figure 13 for a conceptual example of connected components in this setting, where black simply means unconnected. This is only a theory however and needs to be tested for its feasibility.

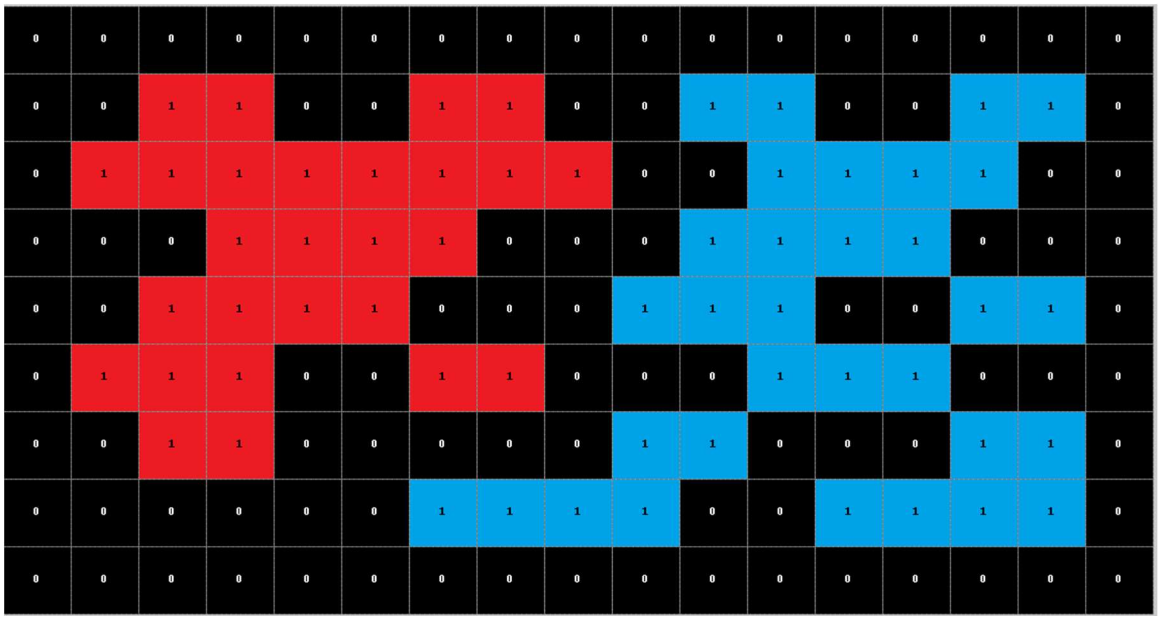


Figure 13. Conceptual connected components results for object detection

FINAL IMPLEMENTATION

This project's research, design and tests have served to identify the requirements for developing a functional implementation of object based classification for FireMAP. Initial development involved using a K-Means algorithm first to get general clusters spectrally grouped and then used DBSCAN to finish spatial clustering. This proved less effective than anticipated, however, because DBSCAN has too much overhead and slow speeds to retain accuracy. This project needs relatively accurate data for grouping pixels into objects such as burn areas, trees, grass, rocks, etc. However, speed is almost more important given the large amount of data in high-resolution imagery. For this reason, efforts went towards developing spatial clustering functionality using connected components because OpenCV already has algorithms for this concept including extrapolating data from those resulting connections. This is important since objects need to have proper information with them such as size, centroid, shape, texture, color, etc. Earlier development and design also looked into using eigenvectors and eigenvalues with PCA algorithms to spectrally cluster objects instead of using K-Means. However, the cost of currently using Eigen functionality is too large given the exponential amount of memory and time required to produce the affinity matrix in high-resolution imagery.

The final implementation of this project loads in an image as a “mat,” which is a native image data type for OpenCV, and then proceeds to use a custom function utilizing OpenCV’s K-Means algorithm (see Appendix A). This process normalizes the image by reducing the variation in colors so that things like shadows or shading through the image do not influence the end result. The optimal “k” value for this kind of imagery is still being researched, but currently it is around five because fire research only really requires detecting areas of ash as well as identifying areas of plant life or lack thereof. This step also does half of the object detection process by grouping pixels based on their color. This is followed by using a custom function to hold an array of the split colors resulting from the previous process (see Appendix B). This step is necessary because OpenCV’s connected components algorithm can only handle binary images. Each split color image has pixels of two values, either with or without color, where each split image is a different color. This becomes clearer once we explore how the connected components aspect of this implementation works. After splitting all the colors into their own images and storing them in an array, one final custom function is used to detect objects spatially by running the connected components algorithm on each of the split images (see Appendix C). When this occurs, connected components searches the image for any pixels next to each other and marks the various groups of them. A connected components ID (cc-id), which identifies spatial clusters within the spectral clusters, is stored in a resulting matrix that keeps track of spatial groups, while a K-Means clustering ID (k-id) unique to each split image is stored in the same matrix and keeps track of which spectral group a pixel belongs to. By combining these two ID's it is possible to search the resulting matrix for unique groups based on color and spatial position. This defines what an object is within an image for this project. The stored spatial and spectral data values make it simple to retrieve any necessary data pertaining to certain pixels or objects. Figures 14-16 provide visual examples of post-fire imagery acquired with a sUAS, where the spatial resolution of this imagery is 6cm, throughout this implementation, though the spatial image is only a data visualization.

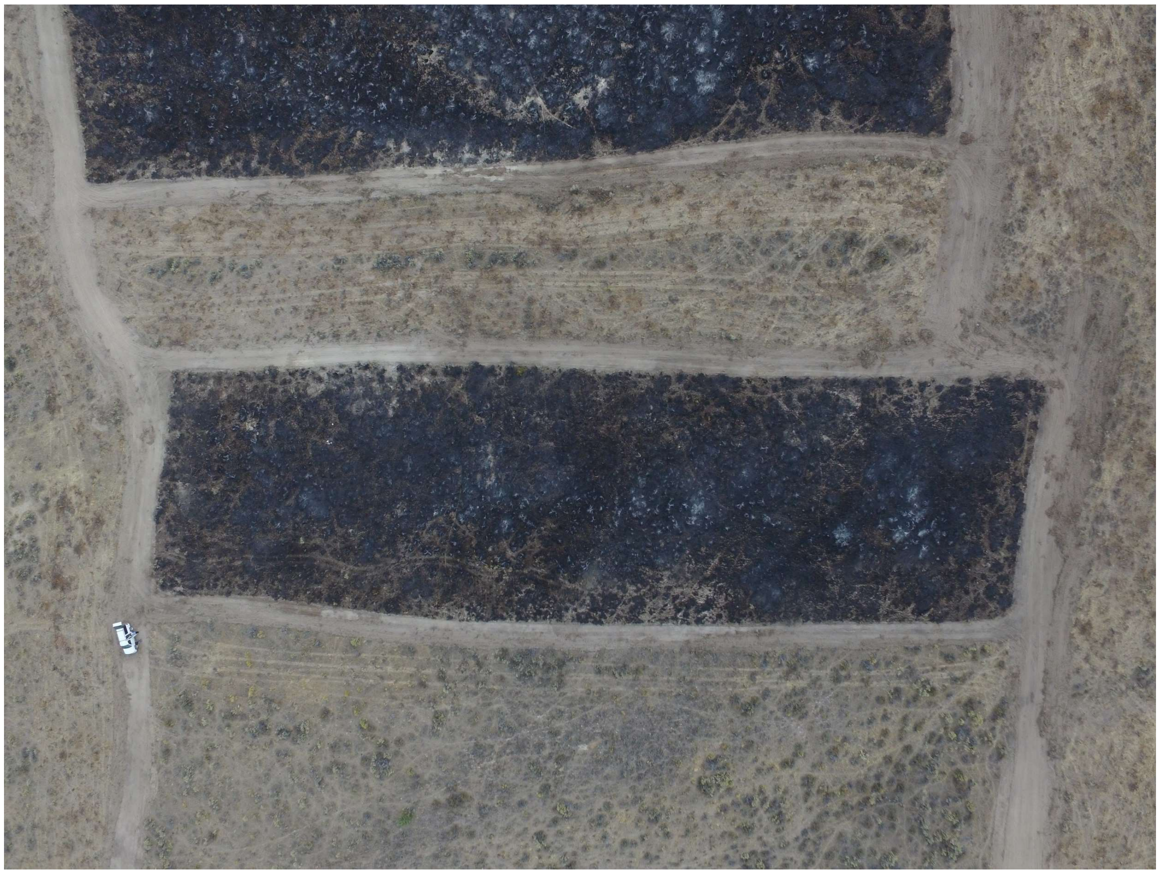


Figure 14. Reynolds creek prescribed burn image

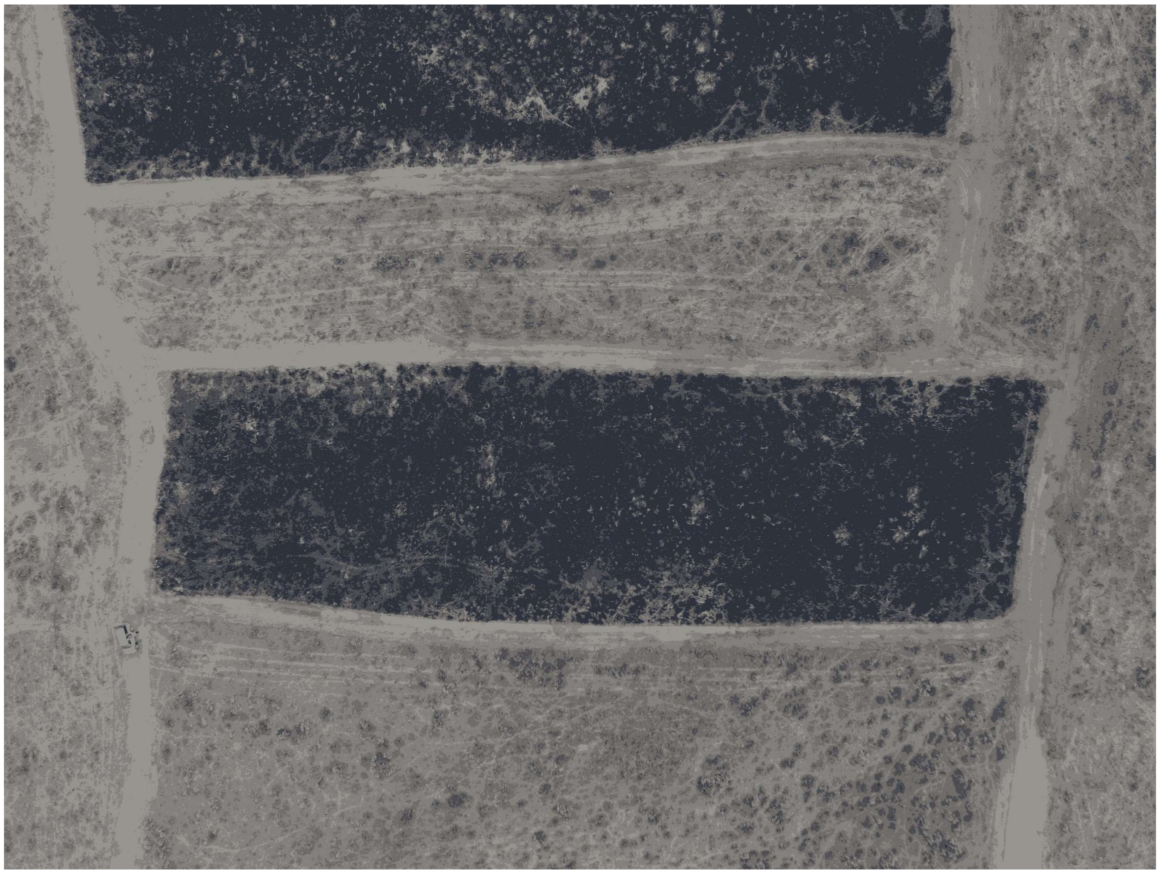


Figure 15. Spectral clustering of Reynolds creek burn

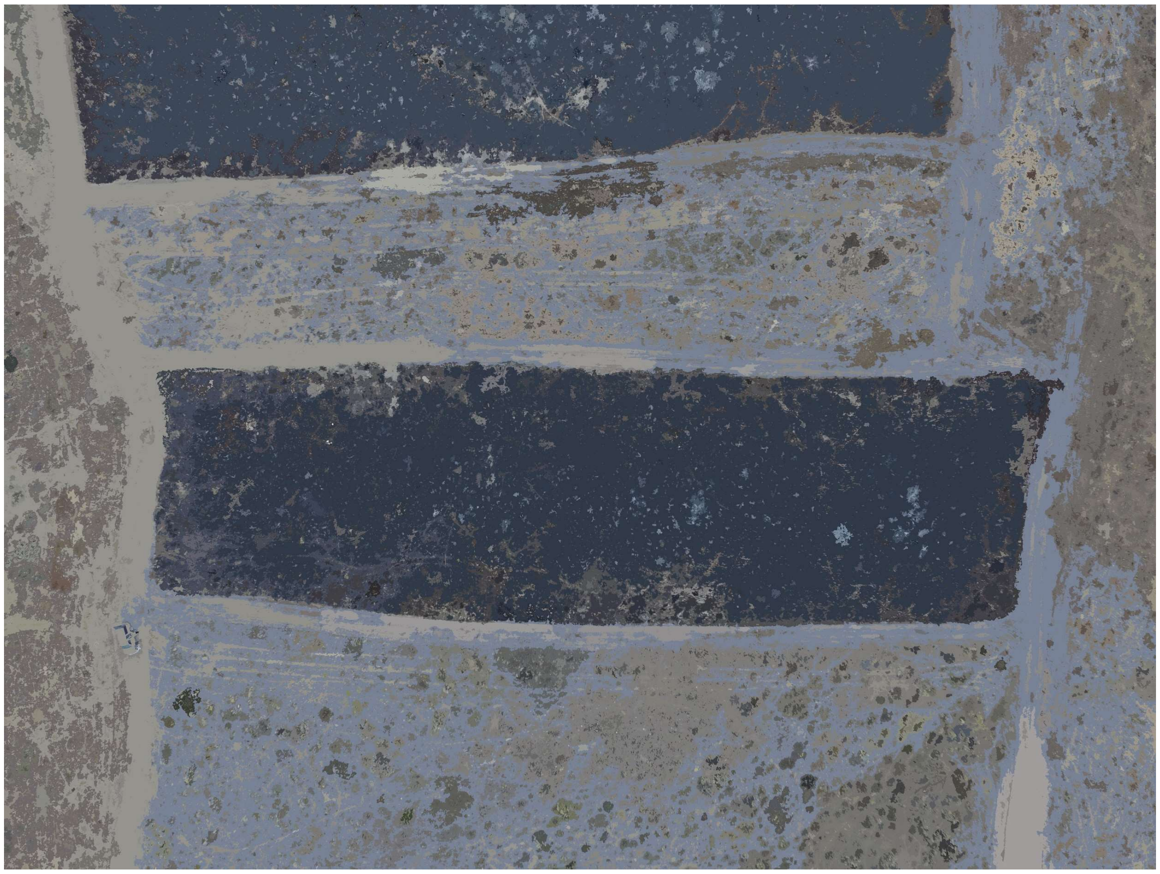


Figure 16. Spatial clustering visualization of Reynolds creek burn

FUTURE WORK

Though this current implementation is functional it is not completely efficient yet. This could be parallelized for faster implementation with the methods described in this paper. The existing algorithm is also rudimentary in that it could be polished with in house code written for algorithms such as the K-Means and connected components. More research is needed for this project to adequately assess what is feasibly the best solution given FireMAP's needs.

CONCLUSION

Many valuable skills were learned during the course of this project. Program development skills were increased by researching problems encountered, coding in C++ both and OpenCV, and researching techniques to find algorithms that could enhance FireMAP. As always the valuable skill of time management is practiced doing a bigger project such as this. It is important to have a schedule and stick to it until it is accomplished, however, when you follow a path that turns out to not be a feasible solution it is wise to keep any research towards that path and to not pursue it further for the time being. Although when it comes to researching and developing such a huge and interesting procedure other responsibilities and distractions can cut into development time, and it is important to make up that work in future free time. Overall this project was a success in several key areas necessary to FireMAP, and through such extensive research, many new avenues and techniques can be considered in other related areas within and out of FireMAP. Spectral and spatial algorithms were researched, designed, and implemented with real test imagery, as well as many other related materials researched for use in this and other projects in the future to further increase efficiency. The resulting approach is functional and the project has the potential to help FireMAP achieve their goal of real-time fire analysis.

Appendix A

```
void KMSC_image(Mat inputImage, Mat &outputImage, int clusters, Mat &labels, Mat
&centers)
{
	// inputImage -	Any acceptable image that OpenCV's Mat can handle.
	// &outputImage	-	Spectrally classified image.
	// clusters   -	Number of clusters desired (K).
	// &labels    -	N by 1 matrix of labels for the classified image.
	(Where N is the number of pixels in the image.)
	// &centers   -	K by CH matrix of the various cluster centroids. (Where
	CH is the number of channels in the image.)

	int R = inputImage.rows;
	int C = inputImage.cols;
	int CH = inputImage.channels();
	int N = R * C;

	/// Reshapes the inputImage, and converts it to datatype float.
	Mat points(N, CH, CV_32F);		// Pixel Matrix, N by CH.

	for (int i = 0; i < R; i++)		// Traverse rows of inputImage.
	{
		for (int j = 0; j < C; j++)		// Traverse columns of inputImage.
		{
			for (int k = 0; k < CH; k++)		// Store values from each
								inputImage channel, k, into
								Pixel Matrix.
			{
				points.at<float>(i + j * R, k) =
inputImage.at<Vec3b>(i, j)[k];		// Vec3b is for 8-bit uchar images with 3
					channels (CV_8UC3).
			}
		}
	}

	/// K-Means clustering of Pixel Matrix to classify inputImage spectrally.
	kmeans(points, clusters, labels, TermCriteria(CV_TERMCRIT_EPS |
CV_TERMCRIT_ITER, 10000, 0.01), 5, KMEANS_PP_CENTERS, centers);

	/// Stores clustered colors to outputImage.
	for (int i = 0; i < R; i++)		// Traverse rows of outputImage.
	{
		for (int j = 0; j < C; j++)		// Traverse columns of
						outputImage.
		{
			int idx = labels.at<int>(i + j * R, 0);		// Index for
									pixel's
									label,
									cluster
									number.
			for (int k = 0; k < CH; k++)
			{
				outputImage.at<Vec3b>(i, j)[k] = centers.at<float>(idx,
k);		// "centers" at cluster number yields that cluster's color.
```

```
                        }
                }
        }
}
```

Appendix B

```
Mat *KMSC_split(Mat inputImage, Mat outputArray[], int clusters, Mat labels)
{
        // inputImage -      Spectrally classified image from KMSC_image.
        // outputArray[]     -      Array of clusters taken from the KMSC_image.
        // clusters   -      Same K value used in the KMSC_image.
        // labels     -      Matrix of labels for the classified image.

        /// Splitting cluster groups into the array.
        for (int x = 0; x < clusters; x++)       // Scan cluster groups.
        {
                Mat init(inputImage.rows, inputImage.cols, CV_8U, Scalar(0));
        // Initialization template for each Mat in array.
                for (int i = 0; i < inputImage.rows; i++)       // Traverse rows of
                                                                inputImage.
                {
                        for (int j = 0; j < inputImage.cols; j++)       // Traverse
                                                                        columns of
                                                                        inputImage.
                        {
                                int idx = labels.at<int>(i + j * inputImage.rows, 0);
        // Index for pixel's cluster number.
                                if (x == idx)       // Pixel found belongs to cluster
                                                    being looked at.
                                {
                                        init.at<uchar>(i, j) = 255;
                                }
                        }
                }
                outputArray[x] = init.clone();
                outputArray[x] = outputArray[x] > 0;            // Conversion to
                                                                Binary.
        }
        return outputArray;
}
```

Appendix C

```
Mat CCOI(Mat inputImage, Mat inputArray[], Mat &outputData, int clusters, Mat
labels)
{
        // inputImage -      Original input image used in KMSC_image.
        // inputArray[]      -      Array of clusters from KMSC_split.
        // &outputData       -      Output Mat of data holding the pixel's object
        IDs.
        // clusters   -      Same K value used in the KMSC_image & KMSC_split.
        // labels     -      Matrix of labels for the classified image.

        Mat outputImage(inputImage.rows, inputImage.cols, inputImage.type(),
Scalar(0, 0, 0));            // Data Visualization



        /// Generate spatial groups from array and store data.
        for (int x = 0; x < clusters; x++)       // Scan through array.
        {
                Mat connections;            // Label for the Connected Components on
                                            each split in the array.
                Mat statistics;             // statistic = stats(label, COLUMN);
                                            where COLUMN is the type of statistic
                                            desired.
                Mat centroids;              // x = centroids(label, 0); y =
                                            centroids(label, 1)

                int rows = inputArray[x].rows;
                int cols = inputArray[x].cols;

                int groups = connectedComponentsWithStats(inputArray[x],
connections, statistics, centroids, 8, CV_32S); /* The original Itype was CV_16U


                                    but if the image is large enough


                                    we can get a group number larger


                                    than the size of a 16 bit integer. */

                for (int i = 0; i < inputArray[x].rows; i++)          // Traverse
                                                                      rows of
                                                                      array[x].

                {
                        for (int j = 0; j < inputArray[x].cols; j++)          //
                                                                      Traverse
                                                                      columns of
                                                                      array[x].

                        {
                                int idxK = labels.at<int>(i + j * inputArray[x].rows,
0);             // K-ID index, where K-ID is the spectral group of the pixel.
                                int idxCC = connections.at<int>(i, j);        // CC-
                                                                      ID index,
                                                                      where CC-ID
                                                                      is the
                                                                      spatial group
                                                                      of the pixel.
                                if (x == idxK)
                                {
                                        /// Store data into outputData.
                                        outputData.at<Vec2i>(i, j)[0] = idxK;
                                        // Store K-ID.
                                        outputData.at<Vec2i>(i, j)[1] = idxCC;
                                        // Store CC-ID.
                                        /// Color outputImage for data visualization. Based on
                                        color from original image.
                                                  int centCol = centroids.at<double>(idxCC, 0);
                                                  int centRow = centroids.at<double>(idxCC, 1);
                                                  outputImage.at<Vec3b>(i, j)[0] =
inputImage.at<Vec3b>(centRow, centCol)[0];
                                                  outputImage.at<Vec3b>(i, j)[1] =
inputImage.at<Vec3b>(centRow, centCol)[1];
                                                  outputImage.at<Vec3b>(i, j)[2] =
inputImage.at<Vec3b>(centRow, centCol)[2];
                                        }
                              }
                    }
          }
          return outputImage;
}
```

Works Cited:

Tan, P., Steinbach, M., & Kumar, V. (2015). Cluster Analysis: Basic Concepts and Algorithms. Introduction to Data Mining (pp. 487-554). Dorling Kindersley: Pearson.

Han, J., Kamber, M., & Pei, J. (2012). Advanced Cluster Analysis. Data Mining: Concepts and Techniques (3rd ed.) (PP. 497-539). Waltham, MA: Morgan Kaufmann.

Salinero, E. C. (2016). Fundamentals of Satellite Remote Sensing: An Environmental Approach. Boca Raton: Taylor & Francis.

Duda, R. O., Hart, P. E., & Stork, D. G. (2001). Introduction. Pattern Classification (pp.1-17). New York: Wiley.

Color quantization. (2017, January 11). Retrieved from https://en.wikipedia.org/wiki/Color_quantization

Hierarchical clustering. (2017, May 05). Retrieved from https://en.wikipedia.org/wiki/Hierarchical_clustering

K-means clustering. (2017, April 20). Retrieved from https://en.wikipedia.org/wiki/K-means_clustering

DBSCAN. (2017, May 04). Retrieved from https://en.wikipedia.org/wiki/DBSCAN

Eigenvalues and eigenvectors. (2017, May 03). Retrieved from https://en.wikipedia.org/wiki/Eigenvalues_and_eigenvectors

OpenCV. Retrieved from http://opencv.org/