Image Anomaly Detection Utilizing the Normalized Laplacian Matrix

Jakob Longbottom

A THESIS

Presented to the Department of Physics
LINFIELD UNIVERSITY
McMinnville, Oregon

In partial fulfillment of the requirements
for the Degree of

BACHELOR OF SCIENCE

May, 2023

Thesis Acceptance

Linfield University

Thesis Title:

Image Anomaly Detection Utilizing the Normalized Laplacian Matrix

Submitted by: Jakob Longbottom

Date Submitted: May, 2023

Thesis Advisor: ______________________________

Dr. Kate Lorenzen

Physics Department: ______________________________

Dr. Michael Crosser

Physics Department: ______________________________

Dr. Joelle Murray

ABSTRACT

Image Anomaly Detection Utilizing the Normalized Laplacian Matrix

Anomaly detection refers to the process of finding objects that do not conform to patterns, such as locating a red balloon amongst blue balloons. The use of algorithms to do this process is becoming more common in practical applications. Through the use of spectral graph theory, the normalized Laplacian matrix, and the exponential distance matrix, this paper extends the work of Verdoja and Grangetto [1] to consider additional algorithms and to compare their effectiveness. Four models were used; including a spectral model, a spatial model (also known as the nearest neighbors), an exponential distance model, and a next-nearest neighbors model. When compared with the given ground truths over real and synthetic anomalies, it was found that the spectral model performed the best. The exponential distance model had similar results to the nearest neighbors model, while the next-nearest neighbors model had the lowest accuracy of outputs.

ACKNOWLEDGEMENTS

Firstly, I would like to thank Dr. Kate Lorenzen, whose expertise and guidance made this project possible. I greatly appreciate the time and advice given through all stages of the project. I would also like to thank the other members of my thesis committee, Dr. Michael Crosser and Dr. Joelle Murray, for their helpful insight and suggestions. My gratitude continues for my fellow classmates and their help in the editing process.

I would also like to recognise the people who have inspired me my entire life. To Mom and Dad, I can never find the words to express the gratitude I feel for being your son. It is because of you both that I am able to be where I am today, and I feel incredibly lucky to have you in my corner.

Finally, I want to give a special thank you to Laney Green. The clock hands never cease to spin, but time stops with you anyway. Thank you for your support and all that you do, I appreciate it more than you know.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The human eye is incredibly keen on detecting patterns. It is able to do so quickly and efficiently. Anomaly detection is the task of finding objects that do not belong based off of the pattern of data.

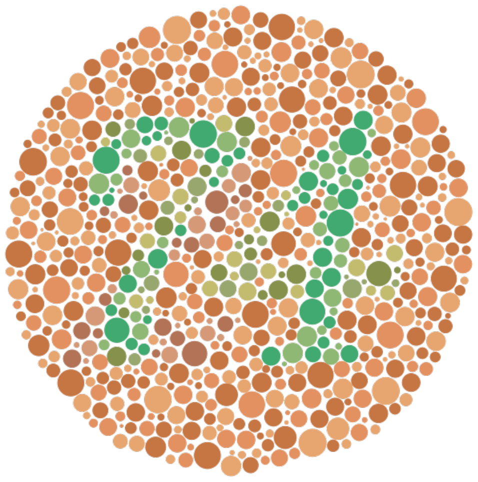


Figure 1.1: Sample of an Ishihara test which is used to test for red-green colorblindness, as the number 74 is not visible to colorblind individuals [2]. Image created by: Shinobu Ishihara, used with permission.

Fig. 1.1 is an example how, assuming one is not red-green colorblind, the human eye is proficient at deciphering patterns. It can be seen in the image the number "74" is displayed through the use of green circles which are the anomalies amongst the red ones. This task is simple for humans, but not as simple when attempted by a computer. Within images, anomalies are a grouping of pixels that are different than those around it. Using Fig. 1.1, a computer could sort through the pixels in the image to find the average color value, the average will be closer to red than green. It could then decide that if a pixel was too far away from the average, such as a green pixel, it would be anomalous. It could then output an image of all the anomalous pixels which would hopefully be in the shape of “74” for the case of Fig. 1.1.

Anomaly detection has been used in image processing for various applications including medical analysis to find tumors in medical scans [1]; video surveillance to detect speeding cars, accidents, or burglaries [3]; and environmental monitoring, such as changes to the landscape or monitoring of development [4]. This paper will consider an application in environmental monitoring as the images used are satellite images looking at buildings in a field.

The Reed-Xiaoli detector (RXD) is considered to be the benchmark algorithm for image anomaly detection [5]. However, this algorithm has drawbacks such as not being able to include spatial information, such as the relation of one pixel to the pixels around it. To try to include this information, Verdoja and Grangetto [1] have used other graph based methods for image anomaly detection. This paper is largely based off of their methods and results and will be referred to throughout. Their methods included the use of two models, written in MATLAB, of a weighted graph, with one model using spectral properties (referring to spectral graph theory and the properties of the eigenvalues of the image’s matrix), this model will be referred to as Model 1. Along with another model using spectral and spatial information, referred to as Model 2, with the spatial information referring to the relation of one pixel to another pixel close by. Both of these models utilize a specialized matrix called the normalized Laplacian matrix which has been shown to be ideal for signal processing; this matrix is discussed in detail in section 2.4.4. The images utilized by Verdoja and

Grangetto [1] will also be used and the statistical analysis conducted uses the human generated ground truths, which were created by them. This was done to allow for direct comparison between papers.

This paper will build off their results by recreating their models, Models 1 and 2, written in Python, with the addition of two new models to compare to their results. The first of the new models, Model 3, is the exponential distance matrix with the nearest neighbors model. The matrix is discussed in section 2.5.1, while the model itself is discussed in section 3.4.1. The final model, Model 4, is the next nearest neighbors model, which looks at spatial information including the pixels a distance of 1 and 2 away from the pixel being analyzed. This model is discussed in section 3.4.2. The analysis at the end of this project revealed that the spectral model performed the best, while the final model, Model 4, performed the worst based off a statistical analysis score, SOI, discussed in Section 2.6.

Throughout this paper certain terms will be used consistently and should be well defined for better understanding. The first term is "anomaly detection", which has been explained above, but will have its definition more formally stated as "the task of spotting items that do not conform to the expected pattern of the data" [1]. The second term is "ground truth", this term will refer to binary masks created by previous researchers [1]. These images will contain the perfect output of an anomaly and will be used as the goal of each model. This is discussed further in section 2.7. It should also be noted that this paper uses the term "spectral" while referring to both color/light in instances of discussing the properties of images. The term is also used to refer to the eigenvalues when discussing the image in a matrix as in spectral graph theory.

Chapter 2 will cover background information for further understanding of hyperspectral bands, spectral graph theory, and the differences between the matrices that were used. Chapter 3 will encompass the methods for all four models used. Chapter 4 will then display the results with analysis and further work being discussed in Chapter 5. The Python code that was used in this project can be found in the Appendix with sections for each model.

Chapter 2

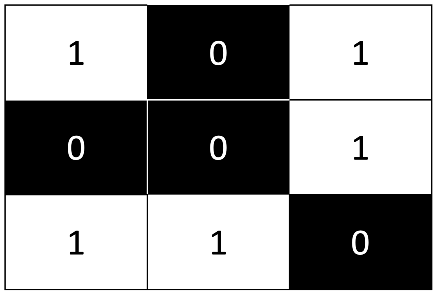
Theory

2.1 Introduction

In order to understand the project as a whole, as well as the methods used within it, there first must be a discussion of the background information. This chapter will cover different topics that will be important to understand for this project. First there will be a discussion of bands in images, building from black and white to hyperspectral bands. Then, there will be some discussion on different matrices and special areas of interest for them. Finally, the concept of ground truth will be explained.

2.2 Pixels with a single value

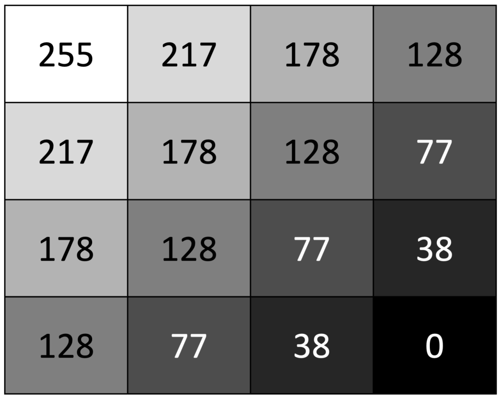
Pixels in images are the small lights that are displayed on a screen in order to create the whole image. There are two types of images where each pixel only has one value: black-and-white images and greyscale images. Black-and-white images are where each pixel is either black or white with no values in between. Greyscale images are where there is some variation between the values allowed for a more gradient flow to occur. There may be some confusion due to the misrepresentation of black and white film, as movies or pictures that are called black and white are actually greyscale.



1	0	1
0	0	1
1	1	0

Figure 2.1: Simplified black and white image containing 9 pixels with either the value of 0 (black) or 1 (white)

In black and white images the pixels have a value of either 0, black, or 1, white. The number can be thought of as the brightness of the pixel. This can be seen in Fig. 2.1, an example of a 9 pixel black and white image.



255	217	178	128
217	178	128	77
178	128	77	38
128	77	38	0

Figure 2.2: An image containing 16 pixels now with values ranging between 255 (white) and 0 (black). The larger the number the darker it is.

Greyscale images allow for the pixels to vary in their brightness. This variation is on a scale from 0, black, to 255, white. The pixel value is stored using an 8-bit integer, which has 256 possibilities. Fig. 2.2 depicts a 16 pixel greyscale image with various values. Since the number is reflective of the brightness of the pixel, the higher the number is the lighter it will be. Greyscale works well for showing some shading in images, but it does not perfectly represent what is shown to the human eye with information about particular colors missing from the images.

2.3 Multiple Bands within Images

Greyscale is nice, but the world is colorful. To represent these colors we need more information. To do so only three values are needed. There are large amounts of combinations that can be made by varying just three values. This is what occurs in televisions, computers, phones, and even our eyes. By looking out into the surroundings there are thousands of colors that can be seen. The human eye can detect light between 380 nm and 700 nm in wavelength. This grouping of light wavelengths is sometimes referred to as the visible light spectrum or the visible light band. A band in light simply refers to a group of wavelengths. The sizing of that band depends on the scenario. For example, the visible light band is between 380 and 700 nm, but within that there are also bands for each visible color, such as the red band from 610 to 700 nm. Pixels in colored images use three bands to represent visible light; Red, Green and Blue (RGB). This was done to recreate how the human eye operates. Within the human eye, specifically the retina, there are cones and rods. Cones are responsible for how the eye perceives color; the human eye has three types of cones. One cone is most sensitive to red light, one is most sensitive to green light, and the other is most sensitive to blue light.

2.3.1 RGB Bands

Visible light can be represented by then increasing and decreasing the amplitudes of each of these bands to represent the color that is desired.

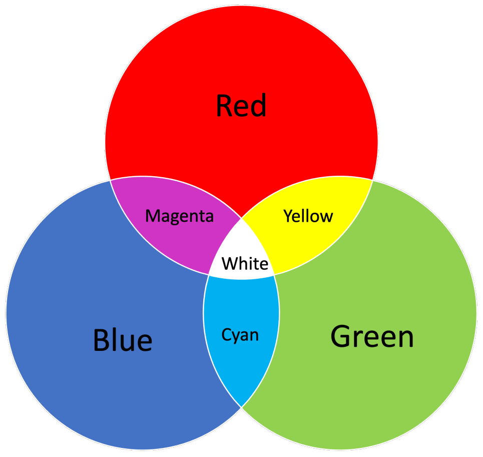


Figure 2.3: RGB color wheel depicting how each of the three primary colors of light interact with one another. When all three are combined in equal amounts white is produced. If only blue and green are combined it will produce cyan, if only blue and red are combined it will produce magenta, and if only red and green are combined it will produce yellow.

Fig. 2.3 shows how each of the three colors combine. The human eye can detect about 100 different shades for each band, leading to around a million different combinations, or colors, that can be made.

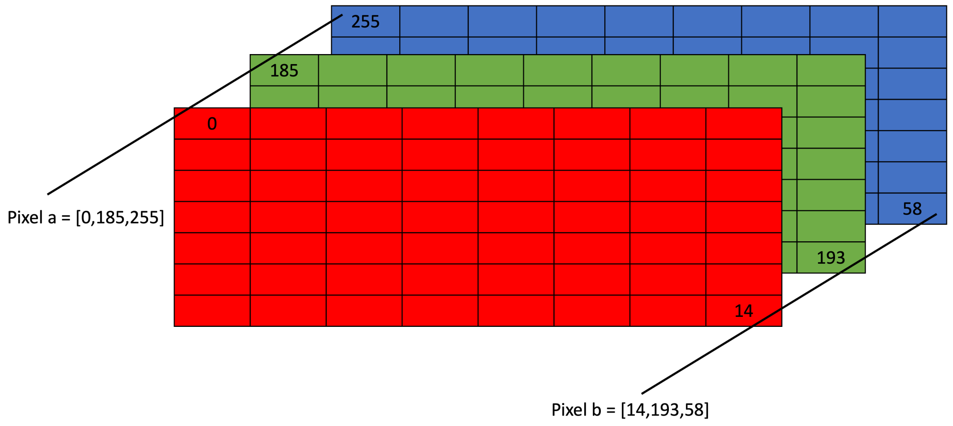


Figure 2.4: In an RGB Array there will be an array corresponding to each color. Each pixel can then be represented with three values, one from each array. Two example pixels are shown, a and b, with their corresponding values shown in the arrays.

To match our perceptions of color, monitors use RGB pixels within an image.

The process of creating different colors is done through the use of arrays. Fig. 2.4 shows how there is an array for each band in RGB, totaling three arrays. A singular pixel will have a value for each one as well. Similar to the greyscale images these pixel values range from 0, no light, to 255, maximum light. This means that a pixel value of [0, 0, 0] will be black and a pixel value of [255, 255, 255] will be white.

2.3.2 Hyperspectral bands

Visible light is only a small section of the electromagnetic spectrum. Electromagnetic waves can vary in wavelength from 0.0001 nm (10^{-4} nm) to 100 m (10^{11} nm). Some of this spectrum can be used as additional information to RGB. This can be input into images using hyperspectral bands, which will include more than just three bands.

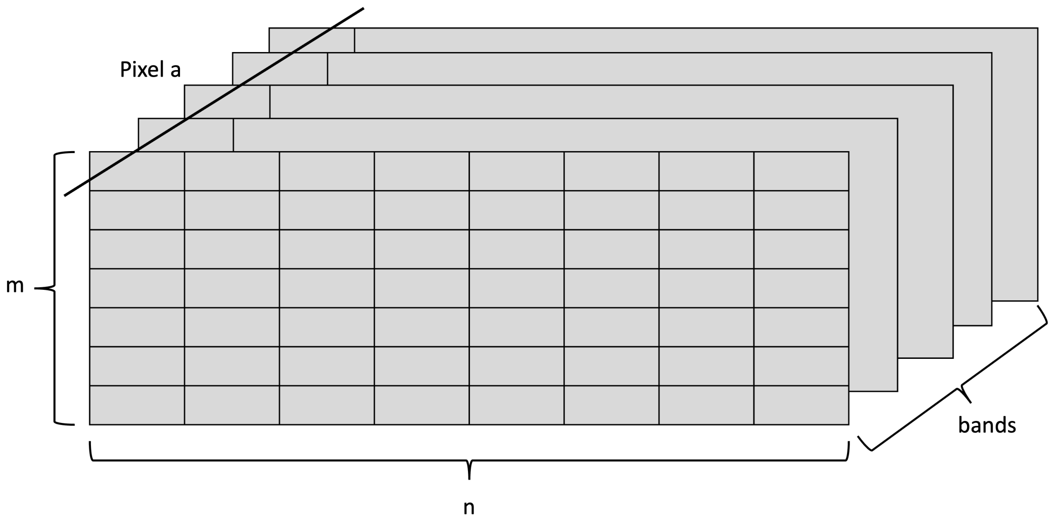


Figure 2.5: Example of a hyperspectral image where m and n are the pixel dimensions. There will one of these sized arrays for each band present. This means that every pixel will have a value for each band as well.

Each band of a hyperspectral image represents a small group of wavelengths. Fig. 2.5 shows how an m by n hyperspectral image will be arranged in a similar manner to RGB images, but with more information. If there are 200 bands then there will be 200 $m \times n$ arrays corresponding to each band.

AVIRIS

The Airborne Visible/Infrared Imaging Spectrometer, AVIRIS, is an optical sensor used for Earth remote sensing by the National Aeronautics and Space Administration, NASA. Its data is mainly used for studying climate change and changes in environment. The instrument has been flown on four aircraft and has flown over the US, Canada, and Europe. During its flight it take images of the landscape using hyperspectral bands.

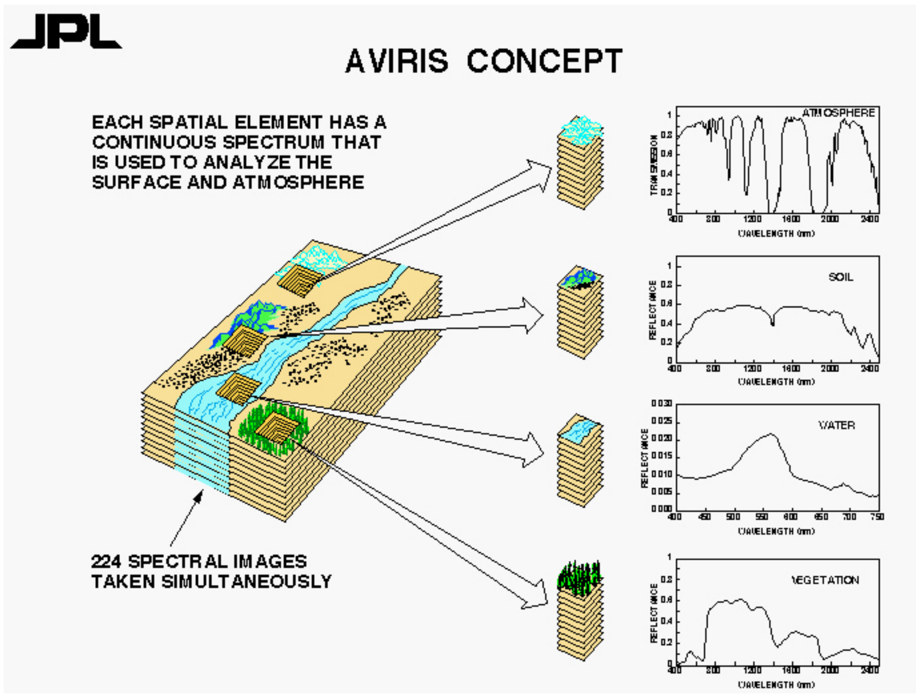

JPL
AVIRIS CONCEPT
EACH SPATIAL ELEMENT HAS A CONTINUOUS SPECTRUM THAT IS USED TO ANALYZE THE SURFACE AND ATMOSPHERE
224 SPECTRAL IMAGES TAKEN SIMULTANEOUSLY
ATMOSPHERE
SOIL
WATER
VEGETATION
TRANSMISSION
REFLECTANCE
WAVELENGTH (nm)

Figure 2.6: The concept of AVIRIS data is displayed. After taking 224 spectral images simultaneously the data can be plotted onto the graph as shown by the right hand side. Each graph is of a single pixel with data from each band being represented.[6] Photo created by Jet Propulsion Labortory, used with permission.

AVIRIS captures images using 224 bands spanning from 380 to 2500 nm wavelengths, with each band having a range of roughly 10 nm. Each of these bands can be correlated with a certain molecular absorption and particle scattering signature [6]. The graphs on the right side of Fig. 2.6 show the spectral data for a single pixel and yields a complete spectrum of visuble, near infrared, and soft wave infrared light (VIS-NIR-SWIR). This can be compared to known VIS-NIR-SWIR spectra of substances to reveal information about its composition. The four graphs in Fig. 2.6 show typical spectra for atmosphere, soil, water, and vegetation.

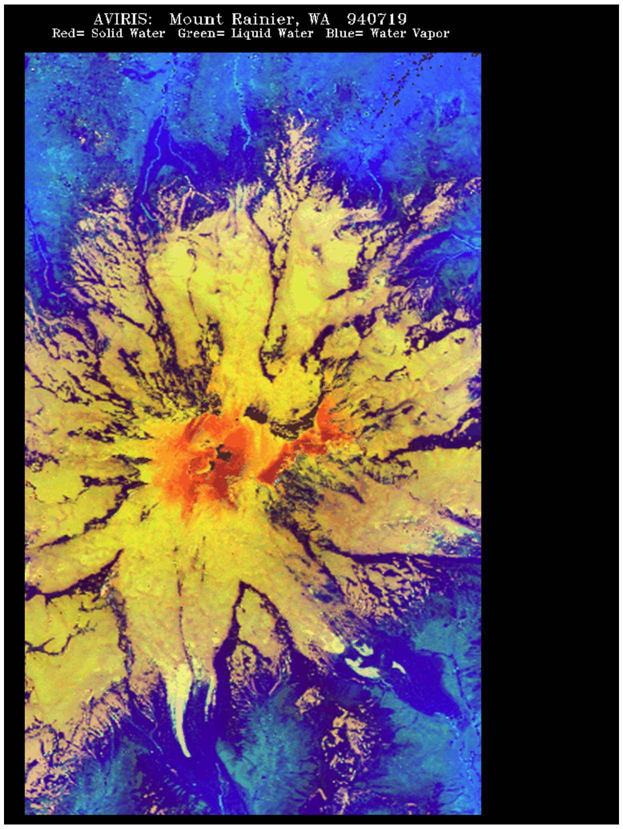

AVIRIS: Mount Rainier, WA 940719
Red= Solid Water Green= Liquid Water Blue= Water Vapor

Figure 2.7: A false color image of the water composition on Mt. Rainier. Red signifies ice, green shows liquid water and blue depicts water vapor. This data was taken on July 14, 1994 [7] Photo taken by AVIRIS, used with permission.

One example of certain bands depicting certain areas of interest is shown in Fig. 2.7. This image shows bands that correlate with water, allowing the viewer to see the different states on Mt. Rainier. There are 20 water bands; bands 108-112, 154-167, 224. This image is a false color image with red showing ice, green showing liquid water, and blue showing water vapor. Yellow light is the combination of red and green light, meaning that the yellow shown in the photo is depicting snow, a mixture between ice and liquid. This shows how these bands can be used to search for something in particular.

2.4 General Graph Theory

A graph $\mathcal{G}$ is a mathematical object that models the connections between objects. Graph $\mathcal{G}$ contains a *vertex* set V($\mathcal{G}$) and an *edge* set E($\mathcal{G}$) . The vertices will represent the objects and the edges will represent the connections between them. If there are two vertices, x and y, an edge between them will be denoted as xy. If there is an xy edge, then it can be said that x and y are adjacent.

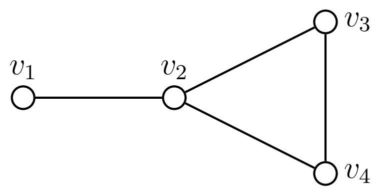


Figure 2.8: Simple paw graph with circles representing the vertices and the lines representing the edges

Figure 2.8 shows a graph with its vertices and edges. It is called a paw graph due to its resemblance to a dog paw. With a small sample size the connectivity of the graph can be seen easily; however, with larger sample sizes these edges can become harder to differentiate from one another. To include more data these graphs can be represented with a matrix. By putting the graph into a matrix it is easier for a computer to use it.

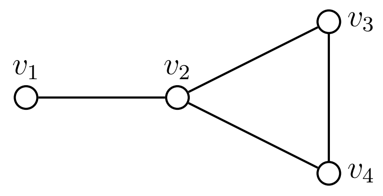
Graph theory is then the study of properties of graphs. Once the graph is represented by a matrix, different information can be studied. Spectral graph theory is the study of the relationship between the graph's structural properties and the matrix's spectral properties such as its eigenvalues and eigenvectors. The structural property depends on the matrix used. This paper will discuss: the adjacency matrix, the degree matrix, the Laplacian matrix, the Normalized Laplacian matrix, the distance matrix, and the exponential distance matrix. For further information on each of the graph matrices see Cvetković *et. al.*'s *Spectra of Graphs* [8]. Several example matrices are defined below.

2.4.1 Adjacency Matrix

To store information about whether two vertices are adjacent (have an edge connecting them) to each other, the adjacency matrix, A, can be utilized by indexing the rows and columns by the vertices of the graph. The *ij*th entry is equal to the number of edges between *i* and *j*. It is defined as follows:

$$A_{ij} = \begin{cases} 1 & \text{if i is adjacent to j} \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

As can be seen from Equation 2.1, the adjacency matrix, A, is defined by values 0 and 1. It will be 1 if there is a connection between the two vertices or they are adjacent to one another. There will be a value of 0 if there is no connection. The adjacency matrix for the graph in Fig. 2.8 is



$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

Figure 2.9: Simple paw graph and its adjacency matrix.

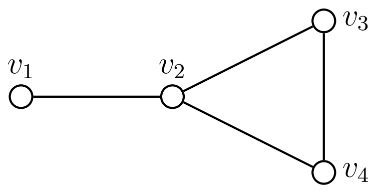
From the matrix it can be seen that vertex 1 is connected to both vertex 2 and 3, but not vertex 4 or itself. This matrix is one more simple way to store the graph.

2.4.2 Degree Matrix

The degree matrix, D, is a diagonal matrix displaying the degrees of each vertex. The degree refers to the number of edges a vertex has.

$$D_{ij} = \begin{cases} deg(v_i) & \text{if i = j} \\ \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

Eq. 2.2 shows how there will only be nonzero values along the diagonal of the matrix. These values will be equal to the degree of vertex i. The degree matrix for Fig. 2.8 is



$$D = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix}$$

Figure 2.10: Simple paw graph and its degree matrix.

In a larger system this matrix is calculated by finding the sum of the columns or rows of the adjacency matrix. Since vertex 1 is only connected to vertex 2, it has the value of 1 in its corresponding index, (1,1).

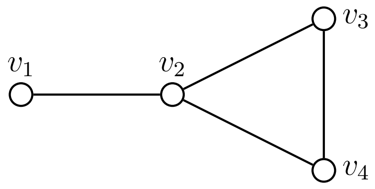
2.4.3 Laplacian Matrix

The Laplacian matrix, L, is defined as the difference between the degree and adjacency matrix,

$$L = D - A \tag{2.3}$$

$$L_{ij} = \begin{cases} deg(v_i) & \text{if i = j} \\ -1 & \text{if i is adjacent to j} \\ 0 & \text{otherwise} \end{cases} \tag{2.4}$$

Eq. 2.3 and Eq. 2.4 show the relationship between the Laplacian matrix and the degree and adjacency matrices. Since the diagonal in the adjacency matrix is filled with zeros and the degree matrix only has nonzero values along the diagonal the combinations is quite simple. The Laplacian matrix combining the matrices 2.9 and 2.10 is



$$L = \begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 3 & -1 & -1 \\ 0 & -1 & 2 & -1 \\ 0 & -1 & -1 & 2 \end{bmatrix}$$

Figure 2.11: Simple paw graph and its Laplacian matrix.

The Laplacian matrix is special in that all of the row sums are equal to zero. This implies that the matrix always has an all ones eigenvector for $\lambda = 0$. The multiplicity of $\lambda = 0$ shows the number of connected components.

2.4.4 Normalized Laplacian Matrix

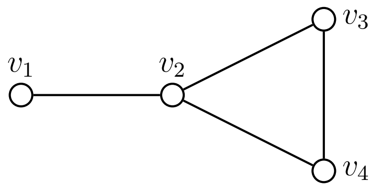
Another matrix is the normalized Laplacian matrix. This matrix is utilized throughout this paper.

$$\mathcal{L} = D^{-1/2}(D - A)D^{-1/2} \tag{2.5}$$

$$\mathcal{L}_{ij} = \begin{cases} 1 & \text{if i = j} \\ \frac{-1}{\sqrt{d_i d_j}} & \text{if i is adjacent to j} \\ 0 & \text{otherwise} \end{cases} \tag{2.6}$$

The normalized Laplacian matrix is defined in Eq. 2.5 with the conditions shown in Eq. 2.6. $D^{-1/2}$ is the inverse square root of the degree matrix. The normalization component coming from these terms as these will make the diagonal entries 1 while preserving the row sums to zero.

The Normalized Laplacian matrix using the graph in Fig. 2.8 is



$$\mathcal{L} = \begin{bmatrix} 1 & \frac{-1}{\sqrt{3}} & 0 & 0 \\ \frac{-1}{\sqrt{3}} & 1 & \frac{-1}{\sqrt{6}} & \frac{-1}{\sqrt{6}} \\ 0 & \frac{-1}{\sqrt{6}} & 1 & \frac{-1}{2} \\ 0 & \frac{-1}{\sqrt{6}} & \frac{-1}{2} & 1 \end{bmatrix}$$

Figure 2.12: Simple paw graph and its Normalized Laplacian matrix.

Certain information should be noted about all of these matrices that give them unique spectral properties. All of these matrices are symmetric, leading to all of their eigenvalues to be real numbers. The normalized Laplacian is special in that its eigenvalues are all in the interval of [0,2]. This fact means that all the entries in the output matrix will also be between 0 and 2, which will help with thresholding, discussed in section 3.5 [9]. The normalized Laplacian matrix also has no isolated vertices.

2.5 Weighted Graphs

Sometimes the connections between vertices are not always equal. One example being if two airports are vertices then the number of flights between them would be the weight. This would allow there to be a differentiation between vertices with strong connections, airports with multiple flights between them, and vertices with weak connections, airports with less flights between them. In these cases the edges are weighted to show the strength of connection. The matrices connected with these kinds of graphs will be similar to that of the unweighted, only instead of an edge being equivalent to 1, it will now be equivalent to the weight associated with the edge.

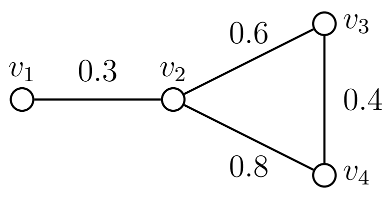


Figure 2.13: Plain graph with weighted edges between the vertices

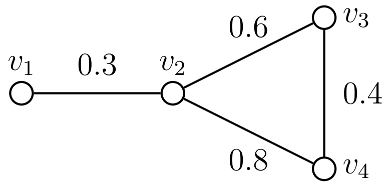
Fig. 2.13 shows the same graph as Fig. 2.8 with the addition of edge weights. The connection between vertex 1 and vertex 2 is 0.3, the connection between vertex 1 and vertex 3 is 0.8, and so on and so forth.

$$A_{ij} = \begin{cases} w_{ij} & \text{if i is adjacent to j} \\ 0 & \text{otherwise} \end{cases} \tag{2.7}$$

Eq. 2.7 shows the conditions of the adjacency matrix. The only change is the substitution of w_{ij} for 1 when the vertices are adjacent.

$$L_{ij} = \begin{cases} -w_{ij} & \text{if i is adjacent to j} \\ \sum_{i=1}^{n} w_{ij} & \text{if i = j} \\ 0 & \text{otherwise} \end{cases} \tag{2.8}$$

Eq. 2.8 shows the conditions of the Laplacian matrix for a weighted graph. The diagonal displays the summation of the weights associated with that vertex, this is equivalent to the degree.



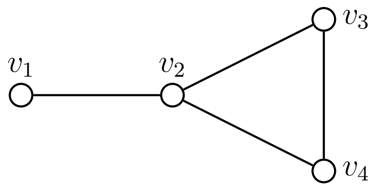
$$L = \begin{bmatrix} 0.3 & -0.3 & 0 & 0 \\ -0.3 & 1.7 & -0.6 & -0.8 \\ 0 & -0.6 & 1 & -0.4 \\ 0 & -0.8 & -0.4 & 1.2 \end{bmatrix}$$

Figure 2.14: The weighted paw graph and its Laplacian matrix.

Here the Laplacian matrix is shown for Figure 2.13. The diagonal shows the summation of the weights along the vertex's edges and the surrounding values display the negative weights of the adjacency matrix.

2.5.1 Distance and Exponential Distance Matrices

The distance between two vertices in a graph is the length of the shortest path between the two vertices, the distance of adjacent vertices is 1. The distance matrix displays these distances; the distance matrix for Fig. 2.8 is



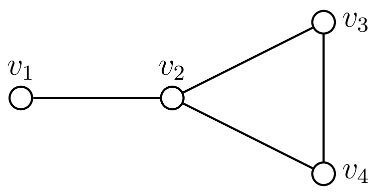
$$dist = \begin{bmatrix} 0 & 1 & 2 & 2 \\ 1 & 0 & 1 & 1 \\ 2 & 1 & 0 & 1 \\ 2 & 1 & 1 & 0 \end{bmatrix}$$

Figure 2.15: Simple paw graph and its distance matrix.

It can be seen how the shortest distances can be calculated for each index. With vertex 2 being a distance of 1 away from all of the other vertices and vertex 1 being a distance of 2 away from vertex 3 and 4.
The exponential distance matrix will be similar but will be defined slightly differently.

$$\mathcal{D}_{q_{ij}} = q^{dist(i,j)} \tag{2.9}$$

where the ith and jth terms are taken as they are from the distance matrix, but now they are put into the exponent of q.



$$\mathcal{D}_{q_{ij}} = \begin{bmatrix} 1 & q & q^2 & q^2 \\ q & 1 & q & q \\ q^2 & q & 1 & q \\ q^2 & q & q & 1 \end{bmatrix}$$

Figure 2.16: Simple paw graph and its exponential distance matrix.

Using Fig. 2.8, Matrix (2.16) is found, q in this example is a variable. Here it can be seen that q^0 will be found along the diagonal, which is equivalent to 1. It should be noted that if q = 1, this becomes an all ones matrix and if q = 0, this becomes the identity matrix. It has been determined that when $|q| < 1$ the matrix is well defined for disconnected graphs [10]. It has been determined for this project for q to equal 1/2.

2.6 Spatial Overlap Index

In order to compare the resulting images to one another statistical analysis is required. This is done using spatial overlap index (SOI) [11].

$$SOI = \frac{2(A \cap B)}{A + B} \tag{2.10}$$

The SOI is a comparison between the resulting image and the hypothetical perfect image. The SOI value can be thought of as the "percent correct". In Equation 2.10, A and B represent the outputted image and the perfect image, respectively. The equation uses the intersection of A and B, the pixels correct, to calculate a ratio for total correct. This will result in a decimal value that will allow for statistical comparison to other models.

$$F_1 = \frac{2 * true\,positive}{2 * true\,positive + false\,positive + false\,negative} \tag{2.11}$$

An F_1-score, Eq. 2.11, is the same equation but wrote out different.

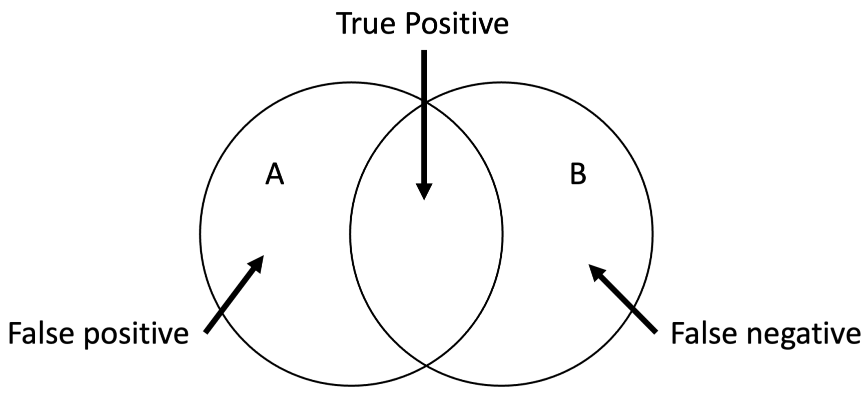
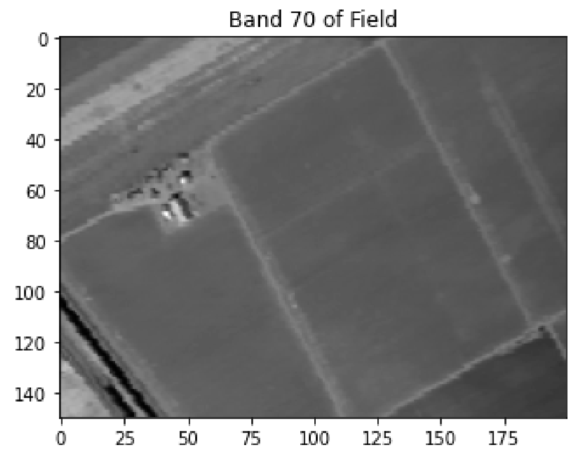


Figure 2.17: SOI is a comparison between two images, A being the resulting image and B being the perfect image.

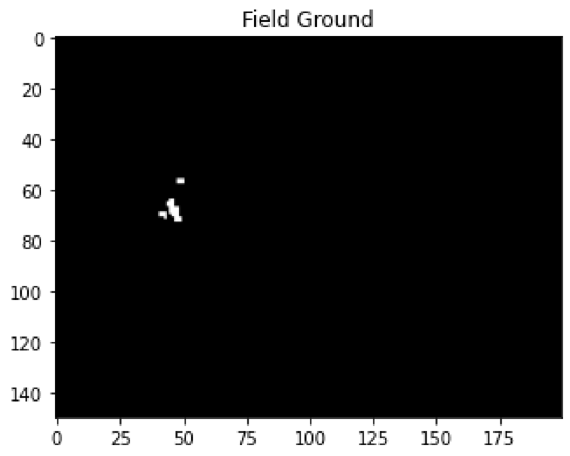
Fig. 2.17 displays the SOI in a more visual way. The true positive is when the pixel is white in both the perfect image and the resulting image. The false positive is when the pixel is white in the resulting image, but is black in the perfect image, or an extra pixel. A false negative is when the pixel is black in the resulting image, but white in the perfect image, or a missed pixel.

2.7 Ground truth

In order to utilize the SOI, a perfect image is needed; this binary scale image will be referred to as the ground truth. For real anomalies, the ground truth is created by a human by going through the image pixel by pixel and selecting which pixels are part of the anomaly. Real anomalies are any natural occurrence of a break in pattern within the image. Synthetic anomalies are anomalies that are inputted into an image by a researcher. This ground truth allows for the actual outputs of the anomaly detection to be compared to the goal of 100% accuracy. It is used to establish how well different techniques work and allow for uniform comparison.



(a) Band 70 of image



(b) Ground truth of the image

Figure 2.18: The same image is displayed with a) band 70 showing what was captured, a house in a field, and b) the ground truth for the anomaly detection of the image

The ground truth of 2.18a can be seen in 2.18b. In this example, the ground truth was a real anomaly so it was created manually.

Synthetic anomalies allow for researchers to identify specific strengths and weaknesses of different models. One way this can be done is through varying the size of the anomaly. This will determine how large an anomaly has to be for the model to capture it. Synthetic anomalies can also vary in their value. This can be done to determine how far away an anomaly has to be from the average value to be considered an anomaly. By having the ability to change the anomaly, researchers can control what information they can infer about the model being tested.

Chapter 3

Methods

This chapter will look at what occurred during the project and how the models were created. For this project there are four models that are compared; a spectral model, a nearest neighbors model, an exponential distance model and a next nearest neighbors model. For this project it was necessary to get the data from the hyperspectral into a 2-dimensional matrix to allow for the calculation of the normalized Laplacian matrix. This matrix was then used to find anomalies within the image by comparing the pixels to the average value of the image. The anomalies were then put through a filter using a matrix creating a black and white picture to compare to the ground truth.

3.1 Initial Setup: Image Flattening

The dataset used for analysis contained 5 images from a previous study [1], these images contained 204 hyperspectral bands and were collected by an AVIRIS sensor. Within this dataset, there were also human generated ground truth images to compare to the final results. These ground truths are discussed in Chapter 2.

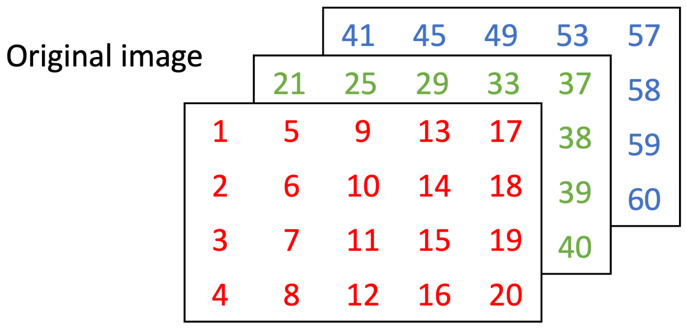


Figure 3.1: A sample 4 × 5 image in RGB is shown. Each pixel has a value corresponding to each band, in this RGB example there are 3 bands so each pixel will have 3 values.

The image is then loaded into Python as a 3-dimensional array, [m,n,z], in a similar way to the sample image shown in Fig. 3.1. For simplicity, examples will be discussed using RGB bands, it should be noted that the project utilized all 204 hyperspectral bands. The first dimension of the array is the number of pixels in a row, the second is the number of pixels in a column, and the third is the number of hyperspectral bands present, in this case 204.

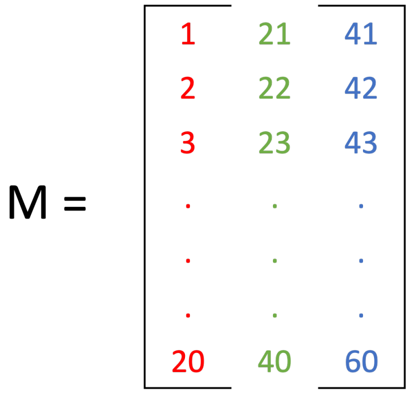


Figure 3.2: Using the same sample image as in Fig. 3.1, the values can then be put into a two dimensional area like the one above. Each column will correspond to an individual band and contain all of its values

This array would then be transformed into a 2 dimensional array, such as the sample matrix shown in Fig. 3.2, with the number of pixels for rows and the bands for columns. This array would then display each row as a singular pixel and each column as a singular band. Now that the 3 dimensional image is translated into a 2 dimensional array, the calculations can begin.

3.2 Spectral Anomaly Detection

The spectral components are the initial building block of all the models are. As can be seen in Fig. 3.3, a single pixel is connected through its spectral bands. Each band is connected to every other band as the sample image displaying red, green, and blue are all connected to one another. It should be noted that the example images will be RGB for simplification, but the images used for this paper contained 204 bands. The 20 water bands were excluded from the total 224 bands that AVIRIS captures, as is common practice [12].

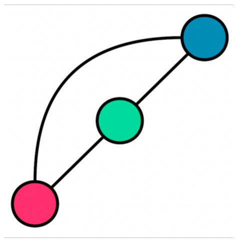


Figure 3.3: Spectral Connectivity of a Pixel: a singuluar pixel is shown with its Red, Green, and Blue bands shown by the circles. The lines between each circle represent each band being connected to the other band.

Referring back to Fig. 3.2, the first step after the set up is to find the average values of each band.

$$\overrightarrow{x} = \begin{bmatrix} \mu_R, & \mu_G, & \mu_B \end{bmatrix} \tag{3.1}$$

A vector could then be created with each average value for each hyperspectral band, as shown in Eq. 3.1. In order to start the set up of the adjacency matrix this vector can be used with its transpose to identify the differences between each band

$$A = |x - x^T| \tag{3.2}$$

Matrix A will create a $z \times z$ matrix. Since Fig. 3.4 is using RGB it is a 3×3 matrix:

$$A = \begin{array}{c|ccc|} & \mu_R & \mu_G & \mu_B \\ \mu_R & 0 & \mu_R - \mu_G & \mu_R - \mu_B \\ \mu_G & \mu_G - \mu_R & 0 & \mu_G - \mu_B \\ \mu_B & \mu_B - \mu_R & \mu_B - \mu_G & 0 \end{array}$$

Figure 3.4: The sample vector from Fig. 3.2 and Eq. 3.2 can be used to create a new matrix. This matrix will then describe the differences between the average values of each band.

However, the images used in this project have 204 bands so they will be 204×204 matrices. The diagonal along A will be 0 as there is no difference between a band and itself. The numbers within the matrix will then represent the difference of averages between the mth and nth bands. A way of adding weights to the matrix is through the use of the Cauchy Distance function [13], which has been shown to capture graph distances effectively for image signals.

$$w_{ab} = \frac{1}{1 + \frac{\mu_a - \mu_b}{\alpha}^2} \tag{3.3}$$

In this function, the w_{ab} term is the weight between band a and band b. μ_a is the average value of band a, with α representing the average value of all of the bands. α is used as a scaling parameter to normalize all the values. This is now the weighted

Adjacency matrix. The degree matrix can be calculated through the column sums of the Adjacency matrix. With both the Adjacency and degree matrices calculated the Normalized Laplacian matrix can be found through Eq. 2.5.

$$\vec{x}^T \mathcal{L} \vec{x} \tag{3.4}$$

The final step is to calculate the anomaly measure for each pixel. Through the use of Eq. 3.4, each pixel can be compared to the average value. This will be done for each individual pixel, resulting in a greyscale image.

3.3 Spatial Components

To further expand off of the spectral model, spatial information, such as using information of the pixels around the pixel in question, can be taken into consideration.

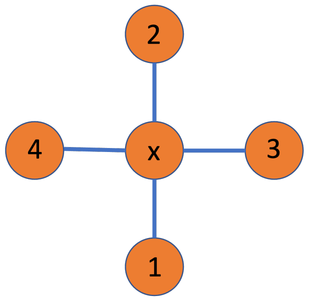


Figure 3.5: Spatial Connectivity of a Pixel: the center pixel (x) is the the pixel of interest with its four nearest neighboring pixels (1-4) also being taken into consideration. The spectral analysis is still being utilized for all of the pixels.

Fig. 3.5 shows how the center pixel, x, is connected to its four nearest neighbors, pixels 1-4. Each one of the pixels will still contain all of its spectral data and will be utilized in a similar way to that of the spectral model.

$$X = \begin{bmatrix} 1 & 21 & 41 & 2 & 22 & 41 & 1 & 21 & 41 & 5 & 25 & 45 & 1 & 21 & 41 \\ 2 & 22 & 42 & 3 & 23 & 42 & 1 & 21 & 41 & 6 & 26 & 46 & 2 & 22 & 42 \\ 3 & 23 & 43 & 4 & 24 & 43 & 2 & 22 & 42 & 7 & 27 & 47 & 3 & 23 & 43 \\ . & . & . & . & . & . & 3 & 23 & 43 & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\ . & . & . & 20 & 40 & 60 & . & . & . & . & . & . & . & . & . \\ 20 & 40 & 60 & 20 & 40 & 60 & 19 & 39 & 59 & 20 & 40 & 60 & 16 & 36 & 56 \end{bmatrix}$$

Figure 3.6: Setting up the 2 dimensional matrix with the inclusion of the spatial connections from the example image in Fig. 3.1. Here the first three columns are the same as those in Fig. 3.2. The next set of three columns refer to pixel 1, with the set of three after that referring to pixel 2, and so on for the other two sets of three columns.

The set up of the initial matrix will be similar to that of the spectral model, as shown in Fig. 3.6. However, this matrix will now include five times as many rows, as there are five total pixels being considered, the center pixel and its 4 nearest neighbors. The second set of three columns displays the information for pixel 1, the pixel below the center pixel. This information will contain the values of the pixel below the values of those to the left of it. As can be seen in the first row, the values are those of the pixel below the pixel in the top right corner of Fig. 3.1. If there is no value for the pixel below it, such as that for pixels on the bottom row, the information will display that of the original pixel, as can be seen in the bottom two rows of the second set of three columns in Fig. 3.6.

$$\vec{x} = \begin{bmatrix} \mu_x, & \mu_x, & \mu_x, & \mu_1, & \mu_1, & \mu_1, & \mu_2, & \mu_2, & \mu_2, & \mu_3, & \mu_3, & \mu_3, & \mu_4, & \mu_4, & \mu_4 \end{bmatrix} \tag{3.5}$$

Following a similar process to the spectral model, a vector, similar to the example vector in Eq. 3.5 can be found displaying the average values for each band. Here the values are then correlated with the band but also the pixel. The example image is RGB so there will be a set of three values for each pixel in the nearest neighbors connection shown in Fig. 3.5.

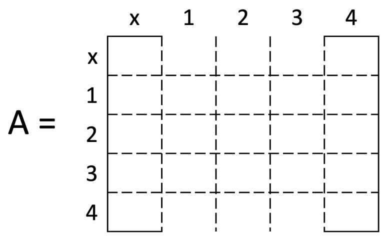


Figure 3.7: The first step of creating the Adjacency matrix can be calculated through the difference of the average value vector, Eq. 3.5, and its transpose. The values on the top and left hand side of the image relate to the connections between the pixels. Within each square there will be a subset matrix.

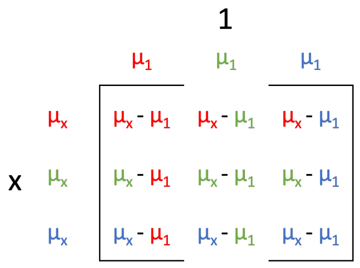


Figure 3.8: This shows an example of the small subset matrix that will go in the x,x slot in Fig. 3.7. Each average value will be compared to every other value in the vector shown in 3.5

Fig. 3.7 shows the first initial set up of the Adjacency matrix. This is calculated through the difference between the average value vector, shown in Eq. 3.5, and its transpose. Since the vector has a length of (pixels considered $\times$ z), the Adjacency matrix will have the size of [pixels considered$\times$z, pixels considered$\times$z]. In the example in Fig. 3.7, there are 5 pixels being considered, the center pixel and its nearest neighbors, and 3 bands, RGB. The result is a [15,15] matrix. Fig. 3.8 shows an example for what is displayed inside each square in the A in Fig. 3.7. This is a [3,3] matrix representing the comparison of the spectral averages of pixel x to itself, This same process will be done to compare all of the pixels to one another.

3.4 Spatial Weights

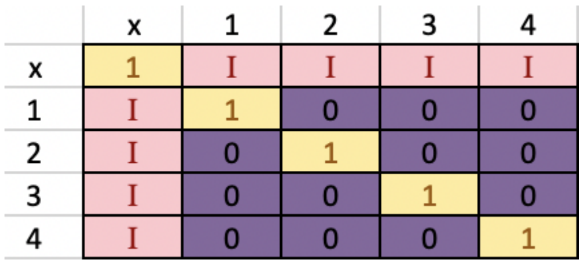
To include the strength of connection between the pixels the adjacency matrix needs to be weighted in a similar manner as the spectral model. The Cauchy Distance function, Eq. 3.3, is used, but this does not result in the final Adjacency matrix like it did in the Spectral model. At this stage the resulting matrix will be referred to the Cauchy Adjacency matrix. The Cauchy Adjacency matrix will need to be multiplied by another weight matrix to take into account the spatial proximity of the pixels.

$$\mathbf{W}(a,b) = \begin{cases} w'_{ab} & \text{if pixels a,b are the same pixel but different bands} \\ w''_{ab} & \text{if pixels a,b are in the same band of nearest neighboring pixels} \\ 0 & \text{otherwise} \end{cases} \tag{3.6}$$

The weight matrix will have the conditions described in Eq. 3.6. It will be the same size as the matrix in Fig. 3.7, [5×z,5×z]. w'_{ab} describes the spectral component and will be equivalent to the Cauchy distance function in Eq. 3.3. This will go along the diagonal of the weight matrix as these areas compare a pixel to itself. w''_{ab} describes the spatial component of the weight matrix. Since it is only when the bands are the same it will only occur on the diagonal of the subset matrices, like that in Fig. 3.8.

$$W = \begin{array}{c} \\ x \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{ccccc} x & 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} w'_{ab} & D_{w''_{ab}} & D_{w''_{ab}} & D_{w''_{ab}} & D_{w''_{ab}} \\ D_{w''_{ab}} & w'_{ab} & D_{w''_{ab}} & D_{w''_{ab}} & D_{w''_{ab}} \\ D_{w''_{ab}} & D_{w''_{ab}} & w'_{ab} & D_{w''_{ab}} & D_{w''_{ab}} \\ D_{w''_{ab}} & D_{w''_{ab}} & D_{w''_{ab}} & w'_{ab} & D_{w''_{ab}} \\ D_{w''_{ab}} & D_{w''_{ab}} & D_{w''_{ab}} & D_{w''_{ab}} & w'_{ab} \end{pmatrix} \end{array} \tag{3.7}$$

In the weight matrix, Eq. 3.7, the rows and columns are shown to be the different pixels (x,1-4) in a similar fashion to Fig. 3.7. Each value inside the matrix will be a subset matrix with the size of [z,z], like that in Fig. 3.8. The w'_{ab} terms represent a full matrix containing off diagonal values as w'_{ab} and the diagonal values as 0 since they represent connection between the same pixel in the same band. The $D_{w''_{ab}}$ terms represent a matrix with w''_{ab} only on the diagonal, as these are the same band. The off diagonal values will be 0.



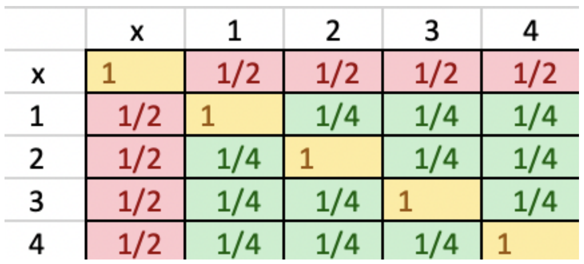
	x	1	2	3	4
x	1	I	I	I	I
1	I	1	0	0	0
2	I	0	1	0	0
3	I	0	0	1	0
4	I	0	0	0	1

Figure 3.9: Weight matrix for the first nearest neighbors model intended to replicate previous results [1]. A "1" represents a [z,z] matrix with all ones on the off diagonal values and 0 on the diagonal. An "I" represents a [z,z] identity matrix, and a "0" represents a [z,z] zero matrix.

To compare against previous works [1], the weight matrix was given weights as shown in Fig. 3.9. Each square is a [z×z] matrix with values corresponding to the value inside them. A "1" represents an all ones off diagonal matrix, an "I" represents an identity matrix, and a "0" represents a zero matrix. This matrix is then multiplied by the Cauchy Adjacency matrix to obtain the weighted adjacency matrix.

3.4.1 Exponential Distance Matrix

To further the analysis, the weights were adjusted to see if there would be any significant improvement in the model. It seems that there should be a connection between pixels 1 and 2. In addition to using the Weight matrix, the exponential distance matrix was utilized.



	x	1	2	3	4
x	1	1/2	1/2	1/2	1/2
1	1/2	1	1/4	1/4	1/4
2	1/2	1/4	1	1/4	1/4
3	1/2	1/4	1/4	1	1/4
4	1/2	1/4	1/4	1/4	1

Figure 3.10: Weight values given to exponential distance matrix. When comparing this weight matrix with the previous one, Fig. 3.9, the “I” is replaced with a “q” and the 0’s are replaced with q^2. Since q was chosen to be equal to 1/2, the matrix will be filled with 1/2 in the first row and column and 1/4 in the rest of the off diagonal indices. The 1/2 and 1/4 represent the values given on the diagonal of the [z,z] matrices they represent.

The third model went through the same steps as described above for spatial information, but instead used the exponential distance matrix for the weight matrix, with its weights being shown in Fig. 3.10. After multiplying this weight matrix by the Cauchy Adjacency matrix the result would be the weighted Adjacency matrix.

3.4.2 Next-Nearest Neighbors

The final model included further spatial information by expanding outward and including information from pixels within two edges of pixel x.

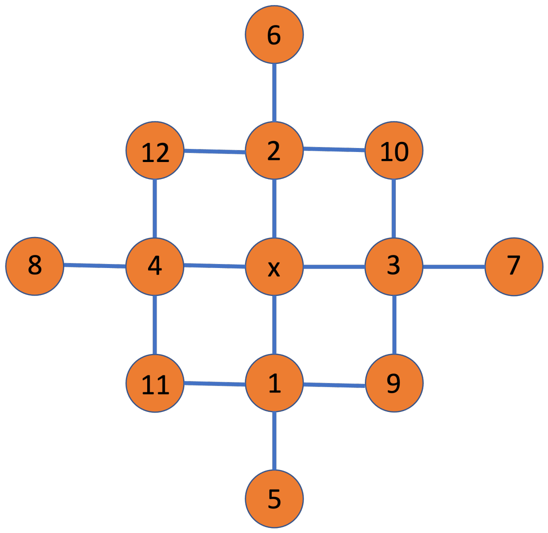
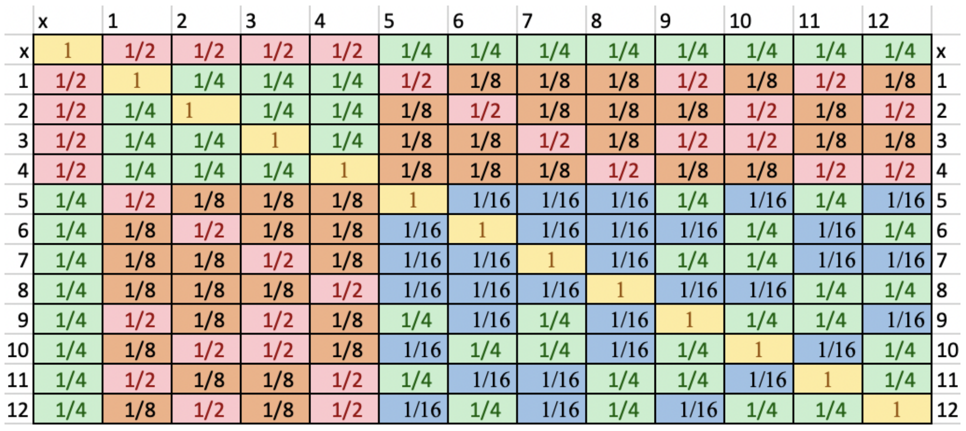


Figure 3.11: The nearest neighbor model, 3.5, can be expanded out further to include the next nearest neighbors as shown above. This includes information from all the pixels within two edges of pixel x.

Fig. 3.11 displays the connections of the next nearest neighbors of pixel x. There are a total of 13 pixels being considered so the Adjacency matrix, as well as the weight matrix, will be a size of [13×z,13×z]. The same steps for the set up are followed with the only difference being the weight matrix.



	x	1	2	3	4	5	6	7	8	9	10	11	12	
x	1	1/2	1/2	1/2	1/2	1/4	1/4	1/4	1/4	1/4	1/4	1/4	1/4	x
1	1/2	1	1/4	1/4	1/4	1/2	1/8	1/8	1/8	1/2	1/8	1/2	1/8	1
2	1/2	1/4	1	1/4	1/4	1/8	1/2	1/8	1/8	1/8	1/2	1/8	1/2	2
3	1/2	1/4	1/4	1	1/4	1/8	1/8	1/2	1/8	1/2	1/2	1/8	1/8	3
4	1/2	1/4	1/4	1/4	1	1/8	1/8	1/8	1/2	1/8	1/8	1/2	1/2	4
5	1/4	1/2	1/8	1/8	1/8	1	1/16	1/16	1/16	1/4	1/16	1/4	1/16	5
6	1/4	1/8	1/2	1/8	1/8	1/16	1	1/16	1/16	1/16	1/4	1/16	1/4	6
7	1/4	1/8	1/8	1/2	1/8	1/16	1/16	1	1/16	1/4	1/4	1/16	1/16	7
8	1/4	1/8	1/8	1/8	1/2	1/16	1/16	1/16	1	1/16	1/16	1/4	1/4	8
9	1/4	1/2	1/8	1/2	1/8	1/4	1/16	1/4	1/16	1	1/4	1/4	1/16	9
10	1/4	1/8	1/2	1/2	1/8	1/16	1/4	1/4	1/16	1/4	1	1/16	1/4	10
11	1/4	1/2	1/8	1/8	1/2	1/4	1/16	1/16	1/4	1/4	1/16	1	1/4	11
12	1/4	1/8	1/2	1/8	1/2	1/16	1/4	1/16	1/4	1/16	1/4	1/4	1	12

Figure 3.12: The next nearest neighbors weight matrix is an expansion of the exponential distance matrix. Using the same q = 1/2, the weights can be calculated for each subset matrix.

The exponential distance matrix is then expanded to include the information from pixels 5-12 to create the next nearest neighbors weight matrix. This can then be applied to the Cauchy Adjacency matrix that has been calculated for this model and the weighted Adjacency matrix for the next nearest neighbors model will be obtained.

The final steps are the same as before. The degree matrix and the Normalized Laplacian matrix are calculated in the same fashion just at a larger scale. The anomaly measure is also calculated with the resulting image being greyscale. The SOI is calculated after comparing the resulting image to the ground truth image.

3.5 Thresholding

To utilize the SOI, however, a black and white image is needed and to do so there needs to be a threshold value to decide what values of the greyscale are white and what values are black. The resulting greyscale image matrix was divided by the maximum value to scale the values down to be between 0 and 1. All of the values now represent the percentage of the maximum value. The image was then looped through each threshold value, from 0 and 1 on a 0.01 increasing factor. If the greyscale pixel value was greater than the threshold value it was given a value of 0 and if it was less than the threshold value it was given a value of 1. This produces a black and white final image to compare to the ground truth. For each threshold value the SOI was calculated. The threshold value with the highest SOI was determined to be the optimal threshold that was then displayed. The SOI value and the threshold value was displayed with the image.

In order to test the SOI calculations within the code, the ground truth is also run through the SOI section. Since the ground truth is being compared to itself, it should result in a 1.00 SOI value for every image.

Chapter 4

Results

This chapter serves as a discussion of the resulting images after running the various models through all the images. A resulting black and white image is displayed depicting the anomaly that was detected. First, the images themselves are compared before looking into the statistical analysis of each image through the use of Spatial Overlap Index (SOI).

4.1 Real Anomalies

Figure 4.1 displays all the images with real anomalies tested. Urban-A, Urban-B, and Field images are used to detect real anomalies. The first row (a-c) depicts a sample band of each image with the band number displayed at the top of each image. These sample bands were chosen to depict how the image is arranged, the models used all the bands in the process of finding the anomalies. The ground truth is displayed below the sample bands and were created by hand, going through the image and selecting the pixels that represented the anomaly [1]. The columns are then arranged by the image that was used. Above each image the t-value is displayed. This is the optimal threshold value that allowed for the best results. Threshold values are discussed in Chapter 3 Methods.

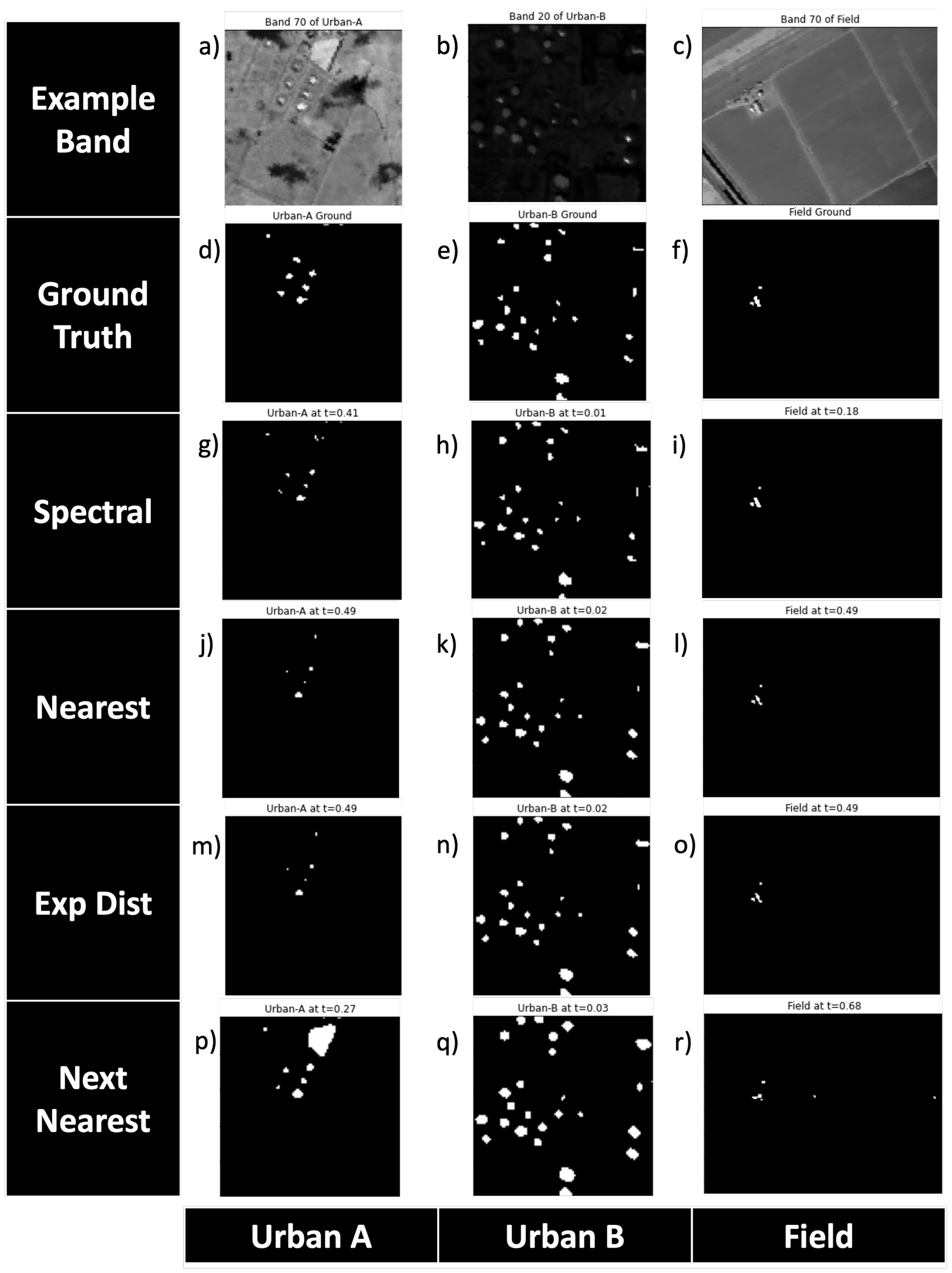


Figure 4.1: Spectral and Hyperspectral Test images for the real anomalies. The example images display a singular band of the images being tested and the ground truth displays the "true" anomaly found by hand[1].

The spectral model (images g,h,i) seems to pick up the most information, but still misses some of image. Comparing the ground truth of Urban A (image d) and the spectral model of Urban A (image g) it can be seen how the top left building of the six block of buildings was not picked up by the spectral model. However, looking at the other models of Urban A, none of them were able to detect this building. When looking at the sample band, that building is not as bright as the other buildings and has less contrast with its surrounding area, leading it to be left out.

When looking at Figure 4.1 it can be seen how in some images the Next Nearest Neighbors model picked up additional pixels compared to the ground truth, such as Urban A, image p. However, for images like Field, image r, the model missed pixels. Image p displays a large mass of pixels in the upper center portion of the image. When looking at the example band, image a, it can be seen that this correlates to what seems to be a large portion of a field or potentially another building. The Field image had similar occurrences. Image r shows a group of pixels both in the center of the image as well as the right side of the image that does not occur in the ground truth, image f. Looking at the sample band, image c, there seem to be some variations of pixels in those areas, but it is difficult to determine what they represent. Further analysis of these examples will be discussed in Chapter 5.

4.2 Synthetic Anomalies

To further test the models, synthetic anomalies are used. Synthetic Light and Synthetic Dark are images that had synthetic anomalies that were manually input into all the bands of the image [1]. The anomalies were created by taking six squares of decreasing size and copying them to make two lines. The second line was then reversed and both lines were then rotated by an angle of $\pi/6$. Two versions of these synthetic anomalies were tested and are named for their outputs of their example band. These are the same images that have been used previously [1]. By varying the size of the squares, the resulting image can then give information on the size of anomalies the model picks up and how precise the model is. The rotation of the sets of squares allows for a better testing of the anomalies, due to the anomalous pixels not being in direct line of one another. This will simulate a real anomaly more accurately as it would be rare to find anomalies in perfect alignment with the image axes.

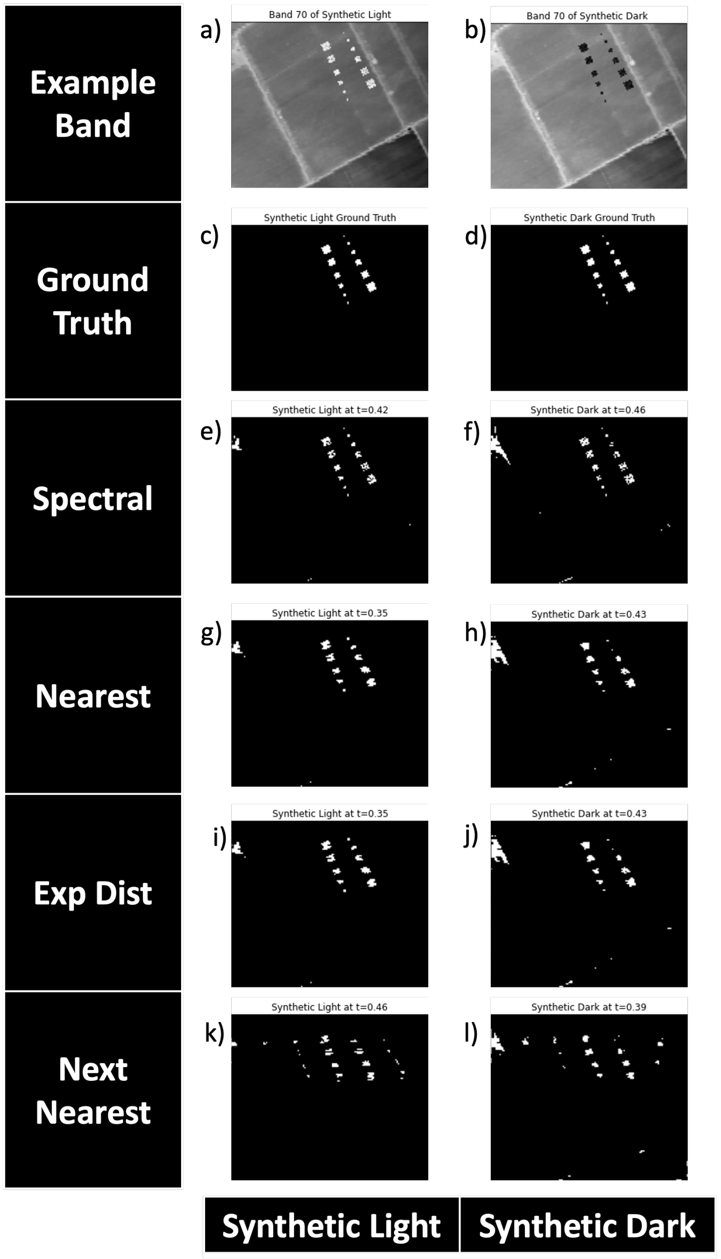


Figure 4.2: Spectral and Hyperspectral Test images for the synthetic anomalies. The example images display a singular band of the images being tested and the ground truth displays the anomaly that was input into all the bands of the image[1].

The synthetic images have similar patterns to the real images in that the next nearest model seems to pick up extra information and the spectral model seems to have the least amount of error. It is interesting to compare the left hand side of the image in all the models. The Synthetic Dark image seems to result in more noise being created by the left hand side when compared with the Synthetic Light counterpart for each model. In the next nearest model there seems to be information that is picked up on both sides of the anomalies in both images k and l.

4.3 SOI comparisons

It can be seen in Figure 4.1 and Figure 4.2 that there are slight differences between each model. However, the scale of the differences is not as simple to figure out simply from the visual components of the images. To analyze the models in a more statistical manner, the spatial overlap index, SOI, values for each image can be analyzed.

Table 4.1: SOI values for all the models tested and for models performed by others [1]. This can be thought of as the percentage of pixels that are correct in the image compared to the ground truth. Averages have been calculated for the real and synthetic images separately as well as together. The total average values for each model are displayed in the bottom row.

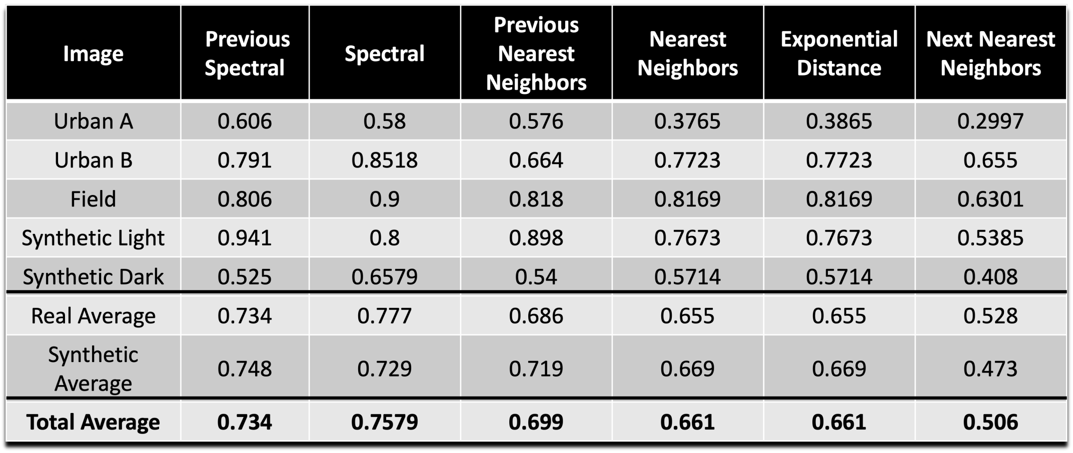


Image	Previous Spectral	Spectral	Previous Nearest Neighbors	Nearest Neighbors	Exponential Distance	Next Nearest Neighbors
Urban A	0.606	0.58	0.576	0.3765	0.3865	0.2997
Urban B	0.791	0.8518	0.664	0.7723	0.7723	0.655
Field	0.806	0.9	0.818	0.8169	0.8169	0.6301
Synthetic Light	0.941	0.8	0.898	0.7673	0.7673	0.5385
Synthetic Dark	0.525	0.6579	0.54	0.5714	0.5714	0.408
Real Average	0.734	0.777	0.686	0.655	0.655	0.528
Synthetic Average	0.748	0.729	0.719	0.669	0.669	0.473
Total Average	**0.734**	**0.7579**	**0.699**	**0.661**	**0.661**	**0.506**

In Table 4.1 the SOI values are shown for each image as well as the average SOI values for each model. The first Spectral column and the first Nearest Neighbors column describe results found by other researchers[1]. The comparison made between image p and image r in Figure 4.1 shows the importance of SOI as it takes both false positives and false negatives into account.

The two spectral models as well as the two nearest neighbors models had similar averages, which was expected. The exponential distance matrix model had no effect on the resulting images, when comparing with the nearest neighbors model. The next nearest neighbors model saw a decrease in SOI values for all of the images.

Going from the Spectral model to the Nearest Neighbors model to the Next Nearest Neighbors model there is a decreasing SOI value for the images. It seems to be that as more information is taken in the more “blurred” the image gets.

Chapter 5

Conclusion

5.1 Reflection on the Project

In conclusion, this thesis has successfully recreated the work done by Verdoja and Grangetto [1], with the addition of two new models. This expansion of models has allowed for further insight, as well as future queries to arise. Looking at the next nearest neighbors model, the model with the largest change, it is interesting to notice the large variation it has within its resulting images. While some of the images seem to pick up more information, other images of the model miss some of the anomalies. Due to the model considering more spatial information it makes sense that small anomalies seem to be overlooked, while the larger anomalies are picked up. This can be seen in images (p) and (r) in Fig. 4.1. It should be noted that the large anomaly mass in image (p) seems to correlate to a large warehouse-like building.

Within the synthetic anomalies the spectral model performed the best, being able to located all 12 anomalies. The nearest neighbors and exponential distance models were able to pick up most of the anomalies, missing the smallest one of each rotated column, which was a size of 12 pixels. The next nearest neighbors model only located the four largest anomalies of each rotated column. This model missed two anomalies, the smallest and the second smallest, which was 25 pixels. The next nearest neighbors model also picked up on a patch of what looks to be a dirt parking lot. This reinforces that the next nearest neighbors model is able to pick up larger anomalies.

One aspect of the project to discuss is the use of optimal thresholding. The resulting images are chosen using the SOI values for each result. This is done to find the optimal output of the model and is done to allow for generalization and flexibility. It did result in a wide range of thresholds, however. The use of the Field image saw a threshold value of 0.18 for the spectral model and a value of 0.68 for the next nearest neighbors model. This is quite a large range for a singular image, and makes it difficult to implement into the real world without a ground truth to compare to. This would need to be looked into further before any applications in the real world.

5.2 Future Work

The spatial and exponential distance model resulting in the same images is peculiar and should be looked into further. The reasoning behind this is still unclear at the time of this writing.

In the Urban-A image used in this paper, there is a large mass that was picked up by the next-nearest neighbors model, but none of the others. This may be due to the optimal thresholding, with other threshold values capturing this data. This mass should be looked into to see if it a natural part of the landscape or if it is in fact an anomaly. If it were to be an anomaly, then a new ground truth should be created for this image that includes the mass.

Due to the next-nearest neighbors model looking at more information, it would be logical to then utilize this model in applications looking for larger anomalies. The previous work that this paper is based on also looked at a medical screening application through the use of fluorodeoxyglucose positron emission tomography (FDG-PET) images [1]. In this application the spatial model outperformed the spectral model, unlike the images used in this project. This could be of interest for the next-nearest neighbors model as this could be beneficial in this application.

Chapter 6

Appendix

6.1 Code for Spectral Model

```
import matplotlib.image as image1
import numpy as np
from scipy.linalg import fractional_matrix_power
from skimage import data
from skimage import color
import matplotlib.pyplot as plt
import scipy.io as sio
urban1 = sio.loadmat('/Users/jakoblongbottom/Desktop/
Phys_Thesis/Phys_490_thesis/Paper_images/urban-1.mat')
#use band 70
urban2 = sio.loadmat('/Users/jakoblongbottom/Desktop/
Phys_Thesis/Phys_490_thesis/Paper_images/urban-2.mat')
#use band 20
impl4 = sio.loadmat('/Users/jakoblongbottom/Desktop/
Phys_Thesis/Phys_490_thesis/Paper_images/
salinas_impl_4.mat')
impl14 = sio.loadmat('/Users/jakoblongbottom/Desktop/
Phys_Thesis/Phys_490_thesis/Paper_images/
salinas_impl_14.mat')
#makes the splotches white instead of black in impl4
field = sio.loadmat('/Users/jakoblongbottom/Desktop/
Phys_Thesis/Phys_490_thesis/Paper_images/salinas.mat')
fieldgt = sio.loadmat('/Users/jakoblongbottom/Desktop/
Phys_Thesis/Phys_490_thesis/Paper_images/salinas_gt.mat')
implgt = sio.loadmat('/Users/jakoblongbottom/Desktop/
Phys_Thesis/Phys_490_thesis/Paper_images/
salinas_impl_gt.mat')

"""
Can change the image used by commenting out current img and
```

```
undoing comment of image desired
"""

img = urban1['data'] #has 204 bands, use 70
imgband = 70
image = "Urban-A"
ground=urban1['map']


"""
img= urban2['data'] #use 20 band
imgband = 20
image = "Urban-B"
ground = urban2['map']
"""

"""
img = impl4['X']
imgband = 70
image = "Synthetic Dark"
ground = implgt['gt']
"""


img = impl14['X']
imgband = 70
image = "Synthetic Light"
ground = implgt['gt']


"""
img = field['X']
imgband = 70
image = "Field"
ground = fieldgt['gt']
"""

plt.title("Band {imgband} of {image}".format(
imgband = imgband,image = image))
plt.imshow(img[:,:,imgband],cmap='gray')
plt.show()


plt.title(image + ' Ground Truth')
plt.imshow(ground,cmap='gray')
plt.show()

#%%
isize = np.size(img[0][0][:]) #number of pixels
isize = int(isize)


x = np.array([])

sizes = img.shape
bands = sizes[2]
X = np.reshape(img,[sizes[0]*sizes[1],bands])

M = X.mean(0)
M = np.reshape(M,(bands,1))

out = np.zeros([sizes[0]*sizes[1]])


Mt = np.transpose(M)

A = abs(M-Mt)

a = np.mean(M)
a = np.reshape(a,(1,1))

A = 1/((1+(A/a))**2)

A = A - np.identity(bands)


D = np.diag(np.sum(A, axis=1))
L = D - A
Dpower = fractional_matrix_power(D,-1/2)
L = Dpower * L * Dpower
end = sizes[0] * sizes[1]

for j in range(1,end):
    x = X[j,:]
    x = np.reshape(x,[bands,1])
    b = x-M
    c = np.matmul(np.transpose(b),np.matmul(L,(b)))
    out[j] = c.real

out = np.reshape(out, [sizes[0],sizes[1]])
```

```
plt.title("{image} Greyscale".format(image = image))
plt.imshow(out, cmap="gray")
plt.show()
#%%
max = np.max(out)
#11.7 million

out = out/ max

#out = ground
#%%
max_thresh_SOI = 0
max_thresh = 0

for i in range(1,100):
    thresh = i/100.0
    binary_mask = out > thresh
    binary_mask = binary_mask*1

    thresh_num = 0
    thresh_denom = 0

    for i in range(0,len(binary_mask)):
        for j in range(0,len(binary_mask[0])):
            if (binary_mask[i][j] == 1 and ground[i][j]==1):
                thresh_num =thresh_num+2
                thresh_denom =thresh_denom+2
            elif (binary_mask[i][j]==1 or ground[i][j] == 1):
                thresh_denom =thresh_denom+1

    thresh_SOI = (thresh_num)/thresh_denom

    if (thresh_SOI > max_thresh_SOI):
        max_thresh_SOI = thresh_SOI
        max_thresh = thresh

print("Spectral Threshold SOI for " + image)
print(np.round(max_thresh_SOI,4))
print( " at ")
print(max_thresh)

binary_mask = out > max_thresh
```

```
fig, ax = plt.subplots()
plt.title("{image} at t={max_thresh}".format(image = image,
max_thresh = max_thresh))
plt.imshow(binary_mask, cmap="gray")
plt.show()
```

6.2 Code for Nearest Neighbors Model

```
import matplotlib.image as image
import numpy as np
from scipy.linalg import fractional_matrix_power
from skimage import data
from skimage import color
import matplotlib.pyplot as plt
import numpy.matlib
import scipy.io as sio
urban1 = sio.loadmat('/Users/jakoblongbottom/Desktop/
Phys Thesis/Phys 490 thesis/Paper images/urban-1.mat')
#use band 70
urban2 = sio.loadmat('/Users/jakoblongbottom/Desktop/
Phys Thesis/Phys 490 thesis/Paper images/urban-2.mat')
#use band 20
impl4 = sio.loadmat('/Users/jakoblongbottom/Desktop/
Phys Thesis/Phys 490 thesis/Paper images/
salinas_impl_4.mat')
impl14 = sio.loadmat('/Users/jakoblongbottom/Desktop/
Phys Thesis/Phys 490 thesis/Paper images/
salinas_impl_14.mat')
#makes the splotches white instead of black in impl4
field = sio.loadmat('/Users/jakoblongbottom/Desktop/
Phys Thesis/Phys 490 thesis/Paper images/salinas.mat')
fieldgt = sio.loadmat('/Users/jakoblongbottom/Desktop/
Phys Thesis/Phys 490 thesis/Paper images/salinas_gt.mat')
implgt = sio.loadmat('/Users/jakoblongbottom/Desktop/
Phys Thesis/Phys 490 thesis/Paper images/
salinas_impl_gt.mat')

"""
Can change the image used by commenting out current img
and undoing comment of image desired
"""


img = urban1['data'] #has 204 bands, use 70
```

```
imgband = 70
image = "Urban-A"
ground=urban1['map']


"""
img= urban2['data'] #use 20 band
imgband = 20
image = "Urban-B"
ground = urban2['map']
"""

"""
img = impl4['X']
imgband = 70
image = "Synthetic Dark"
ground = implgt['gt']
"""


img = impl14['X']
imgband = 70
image = "Synthetic Light"
ground = implgt['gt']


"""
img = field['X']
imgband = 70
image = "Field"
ground = fieldgt['gt']
"""

plt.title("Band {imgband} of {image}".format(
imgband = imgband,image = image))
plt.imshow(img[:,:,imgband])
plt.show()


plt.title(image + ' Ground Truth')
plt.imshow(ground)

isize = np.size(img[0][0][:]) #number of pixels
isize = int(isize)
```

```
#%%

sizes = img.shape
bands = sizes[2]
pixels = sizes[0]*sizes[1]
X = np.reshape(img,[pixels,bands])

i = np.reshape(np.array(range(0,pixels)),[pixels,1])
ii = np.matlib.repmat(i,1,5)

nb = np.reshape(np.array([0,1,-1,sizes[0],-sizes[0]]),[1,5])
#x, right, left, down, up
nbb = np.matlib.repmat(nb,pixels,1)

what = ii+nbb

for k in range(0,pixels):
    if (k % sizes[1] == sizes[1]-1):
        what[k][1] = ii[k][1] #what[k][1] -1

    if ((k % sizes[1]) == 0):
        what[k][2] = ii[k][2] #what[k][2]+1

    if (k >= (pixels-sizes[0])):          #9900:
        what[k][3] = ii[k][3]

    if (k < sizes[0]):
        what[k][4] = ii[k][4]

ii = np.kron(what,np.ones(bands))

j = np.reshape(np.array(range(bands)),[1,bands])
j=(j) * pixels

jj = np.matlib.repmat(j,pixels,5) #this is correct
ij=ii+jj

x = ii

#np.shape(ij)[0]  = pixels
#np.shape(ij)[1] = 1020 (5*bands)

for y in range(np.shape(x)[0]): #pixels
    for z in range(np.shape(x)[1]): #bands
```

```
        band_index = int(ij[y][z]//pixels)
        #could also do z%bands
        pixel_index = int(ij[y][z])%pixels
        x[y][z] = X[pixel_index][band_index]


#%%

out = np.zeros([sizes[0]*sizes[1]])

M = x.mean(0)
M = np.reshape(M,(1,5*bands))

x = x - np.matlib.repmat(M,pixels,1)

Mt = np.transpose(M)

A = abs(M-Mt)

a = np.mean(M)
a = np.reshape(a,(1,1))


#%%

A = 1/((1+(A/a))**2) #cauchy function


#%%
n = 5*bands
W = np.zeros((n,n))
A1 = np.ones((bands,bands))
A2 = np.identity(bands)
W[0:bands,0:n] = np.matlib.repmat(A2,1,5)
W[0:n,0:bands] = np.matlib.repmat(A2,5,1)
W[0:bands,0:bands] = A1

W[(bands):(2*bands), (bands):(2*bands)] = A1
W[(2*bands):(3*bands), (2*bands):(3*bands)] = A1
W[(3*bands):(4*bands), (3*bands):(4*bands)] = A1
W[(4*bands):(5*bands), (4*bands):(5*bands)] = A1


A = A*W
```

```
#%%

A = A - np.identity(5*bands) #no self loops


D = np.diag(np.sum(A, axis=1)) #diagonal matrix of degrees
L = D - A #Laplacian matrix, should be a 1020x1020
Dpower = fractional_matrix_power(D,-1/2)
L = Dpower * L * Dpower #normalized laplacian matrix



for k in range(0,pixels):
    Xx = x[k,:] #all the band values of a single pixel
    Xx = np.reshape(Xx,[5*bands,1])
    # simply getting into column array
    b = Xx
    bt = np.transpose(b)
    c = np.matmul(bt,np.matmul(L,(b)))
    pixel_num = i[k]
    out[pixel_num] = c.real

out = np.reshape(out, [sizes[0],sizes[1]])

#%%
max = np.max(out)
#11.7 million

out = out/ max

#%%
max_thresh_SOI = 0
max_thresh = 0

for k in range(1,100):
    thresh = k/100.0
    binary_mask = out > thresh
    binary_mask = binary_mask*1

    thresh_num = 0
    thresh_denom = 0

    for i in range(0,len(binary_mask)):
        for j in range(0,len(binary_mask[0])):
            if (binary_mask[i][j] == 1 and ground[i][j]==1):
            #true positive
                thresh_num =thresh_num+2
                thresh_denom =thresh_denom+2
            elif (binary_mask[i][j]==1 or ground[i][j] == 1):
            #false positive and false negative
                thresh_denom =thresh_denom+1

    thresh_SOI = (thresh_num)/thresh_denom

    if (thresh_SOI > max_thresh_SOI):
        max_thresh_SOI = thresh_SOI
        max_thresh = thresh

print("Recreation SOI for " + image)
print(np.round(max_thresh_SOI,4))
print( "at threshold of")
print(max_thresh)

binary_mask = out > max_thresh

fig, ax = plt.subplots()
plt.title("{image} at t={max_thresh}".format(image = image,
max_thresh = max_thresh))
plt.imshow(binary_mask, cmap="gray")
plt.show()
```

6.3 Code for Exponential Distance Model

```
import matplotlib.image as image
import numpy as np
from scipy.linalg import fractional_matrix_power
from skimage import data
from skimage import color
import matplotlib.pyplot as plt
import numpy.matlib
import scipy.io as sio
urban1 = sio.loadmat('/Users/jakoblongbottom/Desktop/
Phys Thesis/Phys 490 thesis/Paper images/urban-1.mat')
#use band 70
urban2 = sio.loadmat('/Users/jakoblongbottom/Desktop/
Phys Thesis/Phys 490 thesis/Paper images/urban-2.mat')
#use band 20
impl4 = sio.loadmat('/Users/jakoblongbottom/Desktop/
Phys_Thesis/Phys_490_thesis/Paper_images/
salinas_impl_4.mat')
impl14 = sio.loadmat('/Users/jakoblongbottom/Desktop/
Phys_Thesis/Phys_490_thesis/Paper_images/
salinas_impl_14.mat')
#makes the splotches white instead of black in impl4
field = sio.loadmat('/Users/jakoblongbottom/Desktop/
Phys_Thesis/Phys_490_thesis/Paper_images/salinas.mat')
fieldgt = sio.loadmat('/Users/jakoblongbottom/Desktop/
Phys_Thesis/Phys_490_thesis/Paper_images/salinas_gt.mat')
implgt = sio.loadmat('/Users/jakoblongbottom/Desktop/
Phys_Thesis/Phys_490_thesis/Paper_images/
salinas_impl_gt.mat')
"""
Can change the image used by commenting out current img
and undoing comment of image desired
"""

img = urban1['data'] #has 204 bands, use 70
imgband = 70
image = "Urban-A"
ground=urban1['map']


"""
img= urban2['data'] #use 20 band
imgband = 20
image = "Urban-B"
ground = urban2['map']
"""

"""
img = impl4['X']
imgband = 70
image = "Synthetic Dark"
ground = implgt['gt']
"""


img = impl14['X']
imgband = 70
image = "Synthetic Light"
ground = implgt['gt']
```

```
"""
img = field['X']
imgband = 70
image = "Field"
ground = fieldgt['gt']
"""

plt.title("Band {imgband} of {image}".format(
imgband = imgband,image = image))
plt.imshow(img[:,:,imgband])
plt.show()


plt.title(image + ' Ground Truth')
plt.imshow(ground)
#img = np.transpose(img)

isize = np.size(img[0][0][:]) #number of pixels
isize = int(isize)


#%%

sizes = img.shape
bands = sizes[2]
pixels = sizes[0]*sizes[1]
X = np.reshape(img,[pixels,bands])

i = np.reshape(np.array(range(0,pixels)),[pixels,1])
ii = np.matlib.repmat(i,1,5)

nb = np.reshape(np.array([0,1,-1,sizes[0],-sizes[0]]),[1,5])
#x, right, left, down, up
nbb = np.matlib.repmat(nb,pixels,1)

what = ii+nbb

for k in range(0,pixels):
    if (k % sizes[1] == sizes[1]-1):
        what[k][1] = ii[k][1] #what[k][1] -1

    if ((k % sizes[1]) == 0):
        what[k][2] = ii[k][2] #what[k][2]+1

    if (k >= (pixels-sizes[0])):           #9900:
            what[k][3] = ii[k][3]

        if (k < sizes[0]):
            what[k][4] = ii[k][4]


ii = np.kron(what,np.ones(bands))


j = np.reshape(np.array(range(bands)),[1,bands])
j=(j) * pixels

jj = np.matlib.repmat(j,pixels,5) #this is correct
ij=ii+jj


x = ii


#np.shape(ij)[0]  = pixels
#np.shape(ij)[1] = 1020 (5*bands)


for y in range(np.shape(x)[0]): #pixels
    for z in range(np.shape(x)[1]): #bands
        band_index = int(ij[y][z]//pixels)
        #could also do z%bands
        pixel_index = int(ij[y][z])%pixels
        x[y][z] = X[pixel_index][band_index]


#%%

out = np.zeros([sizes[0]*sizes[1]])

M = x.mean(0)
M = np.reshape(M,(1,5*bands))

x = x - np.matlib.repmat(M,pixels,1)

Mt = np.transpose(M)

A = abs(M-Mt)

a = np.mean(M)
```

```
a = np.reshape(a,(1,1))


#%%

A = 1/((1+(A/a))**2) #cauchy function


#%%
n = 5*bands
W = np.zeros((n,n))
A1 = np.ones((bands,bands))
A2 = np.identity(bands)/2
A3 = A2/2
W[0:bands,0:n] = np.matlib.repmat(A2,1,5)
W[0:n,0:bands] = np.matlib.repmat(A2,5,1)

W[0:bands,0:bands] = A1 #x,x
W[(bands):(2*bands), (bands):(2*bands)] = A1 #1,1
W[(2*bands):(3*bands), (2*bands):(3*bands)] = A1 #2,2
W[(3*bands):(4*bands), (3*bands):(4*bands)] = A1 #3,3
W[(4*bands):(5*bands), (4*bands):(5*bands)] = A1 #4,4



W[(bands):(2*bands), (2*bands):(3*bands)] = A3 #1,2
W[(bands):(2*bands), (3*bands):(4*bands)] = A3 #1,3
W[(bands):(2*bands), (4*bands):(5*bands)] = A3 #1,4

W[(2*bands):(3*bands), (bands):(2*bands)] = A3 #2,1
W[(2*bands):(3*bands), (3*bands):(4*bands)] = A3 #2,3
W[(2*bands):(3*bands), (4*bands):(5*bands)] = A3 #2,4

W[(3*bands):(4*bands), (bands):(2*bands)] = A3 #3,1
W[(3*bands):(4*bands), (2*bands):(3*bands)] = A3 #3,2
W[(3*bands):(4*bands), (4*bands):(5*bands)] = A3 #3,4

W[(4*bands):(5*bands), (bands):(2*bands)] = A3 #4,1
W[(4*bands):(5*bands), (2*bands):(3*bands)] = A3 #4,2
W[(4*bands):(5*bands), (3*bands):(4*bands)] = A3 #4,3


A = A*W

#%%
```

```
A = A - np.identity(5*bands) #no self loops


D = np.diag(np.sum(A, axis=1)) #diagonal matrix of degrees
L = D - A #Laplacian matrix, should be a 1020x1020
Dpower = fractional_matrix_power(D,-1/2)
L = Dpower * L * Dpower #normalized laplacian matrix


for k in range(0,pixels):
    Xx = x[k,:] #all the band values of a single pixel
    Xx = np.reshape(Xx,[5*bands,1])
    # simply getting into column array
    b = Xx
    bt = np.transpose(b)
    c = np.matmul(bt,np.matmul(L,(b)))
    pixel_num = i[k]
    out[pixel_num] = c.real

out = np.reshape(out, [sizes[0],sizes[1]])

#%%
max = np.max(out)
#11.7 million

out = out/ max

#%%
max_thresh_SOI = 0
max_thresh = 0

for k in range(1,100):
    thresh = k/100.0
    binary_mask = out > thresh
    binary_mask = binary_mask*1

    thresh_num = 0
    thresh_denom = 0

    for i in range(0,len(binary_mask)):
        for j in range(0,len(binary_mask[0])):
            if (binary_mask[i][j] == 1 and ground[i][j]==1):
            #true positive
                thresh_num =thresh_num+2
                thresh_denom =thresh_denom+2
            elif (binary_mask[i][j]==1 or ground[i][j] == 1):
            #false positive and false negative
                thresh_denom =thresh_denom+1

    thresh_SOI = (thresh_num)/thresh_denom

    if (thresh_SOI > max_thresh_SOI):
        max_thresh_SOI = thresh_SOI
        max_thresh = thresh

print("Exp Dist SOI for " + image)
print(np.round(max_thresh_SOI,4))
print( " at threshhold of ")
print(max_thresh)

binary_mask = out > max_thresh

fig, ax = plt.subplots()
plt.title("{image} at t={max_thresh}".format(image = image,
max_thresh = max_thresh))
plt.imshow(binary_mask, cmap="gray")
plt.show()
```

6.4 Code for Next Nearest Neighbors Model

```
import matplotlib.image as image
import numpy as np
from scipy.linalg import fractional_matrix_power
from skimage import data
from skimage import color
import matplotlib.pyplot as plt
import numpy.matlib
import scipy.io as sio
urban1 = sio.loadmat('/Users/jakoblongbottom/Desktop/
Phys Thesis/Phys 490 thesis/Paper images/urban-1.mat')
#use band 70
urban2 = sio.loadmat('/Users/jakoblongbottom/Desktop/
Phys Thesis/Phys 490 thesis/Paper images/urban-2.mat')
#use band 20
impl4 = sio.loadmat('/Users/jakoblongbottom/Desktop/
Phys Thesis/Phys 490 thesis/Paper images/
salinas_impl_4.mat')
impl14 = sio.loadmat('/Users/jakoblongbottom/Desktop/
Phys_Thesis/Phys_490_thesis/Paper_images/
salinas_impl_14.mat')
#makes the splotches white instead of black in impl4
field = sio.loadmat('/Users/jakoblongbottom/Desktop/
Phys_Thesis/Phys_490_thesis/Paper_images/salinas.mat')
fieldgt = sio.loadmat('/Users/jakoblongbottom/Desktop/
Phys_Thesis/Phys_490_thesis/Paper_images/salinas_gt.mat')
implgt = sio.loadmat('/Users/jakoblongbottom/Desktop/
Phys_Thesis/Phys_490_thesis/Paper_images/
salinas_impl_gt.mat')
"""
Can change the image used by commenting out current img
and undoing comment of image desired
"""


img = urban1['data'] #has 204 bands, use 70
imgband = 70
image = "Urban-A"
ground=urban1['map']


"""
img= urban2['data'] #use 20 band
imgband = 20
image = "Urban-B"
ground = urban2['map']
"""

"""
img = impl4['X']
imgband = 70
image = "Synthetic Dark"
ground = implgt['gt']
"""



img = impl14['X']
imgband = 70
image = "Synthetic Light"
ground = implgt['gt']


"""
```

```
img = field['X']
imgband = 70
image = "Field"
ground = fieldgt['gt']
"""

plt.title("Band {imgband} of {image}".format(
imgband = imgband,image = image))
plt.imshow(img[:,:,imgband])
plt.show()


plt.title(image + ' Ground Truth')
plt.imshow(ground)

isize = np.size(img[0][0][:]) #number of pixels
isize = int(isize)


#%%

sizes = img.shape
bands = sizes[2]
pixels = sizes[0]*sizes[1]
X = np.reshape(img,[pixels,bands])

#%%


i = np.reshape(np.array(range(0,pixels)),[pixels,1])
ii = np.matlib.repmat(i,1,13)

nb = np.reshape(np.array([0,sizes[0],-sizes[0],1,-1,
2*sizes[0],-2*sizes[0],2,-2,
sizes[0]+1, -sizes[0]+1, sizes[0]-1,-sizes[0]-1]),[1,13])
 #x, down, up, right, left,
 #2 down, 2 up, 2 right, 2 left,
 #downright, upright, downleft, upleft
nbb = np.matlib.repmat(nb,pixels,1)

what = ii+nbb

"""
bands = 3
pixels = 100
```

```
sizes=[10,10]
"""

for k in range(0,pixels):
    if (k >= (pixels-sizes[0])): #down
        what[k][1] = ii[k][1]
        #what[k][9] = ii[k][9]
        #what[k][11] =ii[k][11]

    if (k < sizes[0]):             #up
        what[k][2] = ii[k][2]
        #what[k][10] = ii[k][10]
        #what[k][12] = ii[k][12]

    if (k % sizes[1] == sizes[1]-1):      #right
        what[k][3] = ii[k][3]
        #what[k][9] = ii[k][9]
       # what[k][10] = ii[k][10]

    if ((k % sizes[1]) == 0):          #left
        what[k][4] = ii[k][4]
        #same as before inner 4 pixels
        #what[k][11] = ii[k][11]
        #what[k][12] = ii[k][12]

    if (k < 2*sizes[0]):           #2 up
        what[k][6] = ii[k][6]

    if (k >= (pixels-2*sizes[0])):   #2 down
        what[k][5] = ii[k][5]

    if (k%sizes[1]>= sizes[1] -2):         #2 right
        what[k][7] = ii[k][7]

    if (k%sizes[1] <= 1):              #2 left
        what[k][8] = ii[k][8]

    if (k % sizes[1]== sizes[1]-1 or k >= (pixels-sizes[0])):
        what[k][9] = ii[k][9]

    if (k<sizes[0] or k % sizes[1] == sizes[1]-1):
        what[k][10] = ii[k][10]

    if (k >= (pixels-sizes[0]) or (k % sizes[1] == 0)):
        what[k][11] = ii[k][11]
```

```
          if (k<sizes[0] or (k % sizes[1]) == 0):
              what[k][12] = ii[k][12]


    ii = np.kron(what,np.ones(bands))


    j = np.reshape(np.array(range(bands)),[1,bands])
    j=(j) * pixels

    jj = np.matlib.repmat(j,pixels,13) #this is correct
    ij=ii+jj


    #%%

    x = ii

    #np.shape(ij)[0]  = pixels
    #np.shape(ij)[1] = 2652 (13*bands)


    for y in range(np.shape(x)[0]): #pixels
        for z in range(np.shape(x)[1]): #bands
            #band_index = int(ij[y][z]//pixels)
            #could also do z%bands
            pixel_index = int(ij[y][z])%pixels
            x[y][z] = X[pixel_index][z%bands]

    #%%

    out = np.zeros([sizes[0]*sizes[1]])

    M = x.mean(0)
    M = np.reshape(M,(1,13*bands))

    x = x - np.matlib.repmat(M,pixels,1)

    Mt = np.transpose(M)

    A = abs(M-Mt)

    a = np.mean(M)
    a = np.reshape(a,(1,1))
```

```
A = 1/((1+(A/a))**2) #cauchy function


#%%
n = 13*bands
W = np.zeros((n,n))
A1 = np.ones((bands,bands))
A2 = np.identity(bands)/2 #1/2
A3 = A2/2 #1/4
A4 = A3/2 #1/8
A5 = A4/2 #1/16

#x-4 pixels
W[0:5*bands,0:5*bands] = np.matlib.repmat(A3,5,5)


#pixels related to x on x axis or top row
W[0:bands,bands:5*bands] = np.matlib.repmat(A2,1,4)
# 1-4, 0
W[0:bands,5*bands:13*bands] = np.matlib.repmat(A3,1,8)
#5-12,0

#pixels related to x on y axis or first column
W[bands:5*bands, 0: bands] = np.matlib.repmat(A2,4,1)
#0,1-4
W[5*bands:13*bands, 0: bands] = np.matlib.repmat(A3,8,1)
#0,5-12


#bottom left square
W[5*bands:13*bands,bands:5*bands]=np.matlib.repmat(A4,8,4)
#all the 1/8s
for k in range(1,4):
     W[(k+4)*bands:(k+5)*bands,k*bands:(k+1)*bands] = A2
     #5,1/ 6,2 / 7,3 / 8,4
W[9*bands:10*bands,bands:2*bands] = A2      #9,1
W[9*bands:10*bands,3*bands:4*bands] = A2    #9,4
W[10*bands:11*bands,2*bands:4*bands]=np.matlib.repmat(A2,1,2)
#10,2 and 3
W[11*bands:12*bands,bands:2*bands] = A2      #11,1
W[12*bands:13*bands,2*bands:3*bands] = A2 #12,2
W[11*bands:13*bands,4*bands:5*bands]=np.matlib.repmat(A2,2,1)
#11 and 12, 4
```

```
#top right square
W[bands:5*bands,5*bands:13*bands]=np.matlib.repmat(A4,4,8)
#all the 1/8s
for k in range(1,4):
    W[(k)*bands:(k+1)*bands,(k+4)*bands:(k+5)*bands] = A2
    #1,5 / 2,6 / 3,7 / 4,8
W[1*bands:2*bands,9*bands:10*bands] = A2 #1,9
W[1*bands:2*bands,11*bands:12*bands] = A2 #1,11
W[3*bands:4*bands,9*bands:10*bands] = A2 #3,9
W[4*bands:5*bands,11*bands:13*bands]=np.matlib.repmat(A2,1,2)
#4,11 and 12
W[2*bands:4*bands,10*bands:11*bands]=np.matlib.repmat(A2,2,1)
#2 and 3 , 10
W[2*bands:3*bands,12*bands:13*bands] = A2 #2,12


#bottom right square
W[5*bands:13*bands,5*bands:13*bands] =
np.matlib.repmat(A5,8,8) #all the 1/16s
W[9*bands:10*bands,5*bands:6*bands] = A3 #9,5
W[11*bands:12*bands,5*bands:6*bands] = A3 #11,5
W[10*bands:11*bands,6*bands:7*bands] = A3 #10,6
W[12*bands:13*bands,6*bands:7*bands] = A3 #12,6
W[9*bands:11*bands,7*bands:8*bands] =
np.matlib.repmat(A3,2,1) #9 and 10,7
W[11*bands:13*bands,8*bands:9*bands] =
np.matlib.repmat(A3,2,1) #11 and 12,8

W[10*bands:12*bands,9*bands:10*bands] =
np.matlib.repmat(A3,2,1) #10 and 11,9
W[5*bands:6*bands,9*bands:10*bands] = A3 #5,9
W[7*bands:8*bands,9*bands:10*bands] = A3 #7,9

W[6*bands:8*bands,10*bands:11*bands] =
np.matlib.repmat(A3,2,1) #6 and 7,10
W[9*bands:10*bands,10*bands:11*bands] = A3 #9,10
W[12*bands:13*bands,10*bands:11*bands] = A3 #12,10

W[8*bands:10*bands,11*bands:12*bands] =
np.matlib.repmat(A3,2,1) #8 and 9,11
W[5*bands:6*bands,11*bands:12*bands] = A3 #5,11
W[12*bands:13*bands,11*bands:12*bands] = A3 #12,11

W[10*bands:12*bands,12*bands:13*bands] =
np.matlib.repmat(A3,2,1) #10 and 11,12
W[6*bands:7*bands,12*bands:13*bands] = A3 #6,12
W[8*bands:9*bands,12*bands:13*bands] = A3 #8,12


for k in range(0,13):
    W[k*bands:(k+1)*bands,k*bands:(k+1)*bands] = A1 #x,x

#%%

A = A*W
A = A - np.identity(13*bands) #no self loops


D = np.diag(np.sum(A,axis=1)) #diagonal matrix of degrees
L = D - A #Laplacian matrix, should be a 1020x1020
Dpower = fractional_matrix_power(D,-1/2)
L = Dpower * L * Dpower #normalized laplacian matrix

for k in range(0,pixels):
    Xx = x[k,:] #all the band values of a single pixel
    Xx = np.reshape(Xx,[13*bands,1])
    # simply getting into column array
    b = Xx
    bt = np.transpose(b)
    c = np.matmul(bt,np.matmul(L,(b)))
    pixel_num = i[k]
    out[pixel_num] = c.real

out = np.reshape(out, [sizes[0],sizes[1]])

#%%
max = np.max(out)
#11.7 million

out = out/ max

#%%
max_thresh_SOI = 0
max_thresh = 0

for k in range(1,100):
    thresh = k/100.0
    binary_mask = out > thresh
    binary_mask = binary_mask*1
    thresh_num = 0
    thresh_denom = 0

    for i in range(0,len(binary_mask)):
        for j in range(0,len(binary_mask[0])):
            if (binary_mask[i][j] == 1 and ground[i][j]==1):
            #true positive
                thresh_num =thresh_num+2
                thresh_denom =thresh_denom+2
            elif (binary_mask[i][j]==1 or ground[i][j] == 1):
            #false positive and false negative
                thresh_denom =thresh_denom+1

    thresh_SOI = (thresh_num)/thresh_denom

    if (thresh_SOI > max_thresh_SOI):
        max_thresh_SOI = thresh_SOI
        max_thresh = thresh

print("Next Nearest SOI for " + image)
print(np.round(max_thresh_SOI,4))
print( " at threshhold of ")
print(max_thresh)

binary_mask = out > max_thresh

fig, ax = plt.subplots()
plt.title("{image} at t={max_thresh}".format(image = image,
max_thresh = max_thresh))
plt.imshow(binary_mask, cmap="gray")
plt.show()
```

References

[1] F. Verdoja and M. Grangetto, Machine Vision and Applications **31**, 11 (2020).

[2] Shinobu Ishihara, Ishihara Plate 9, vectorized, 2021.

[3] K.-W. Cheng, Y.-T. Chen, and W.-H. Fang, IEEE Transactions on Image Processing **24**, 5288 (2015).

[4] D. J. Hill and B. S. Minsker, Environmental Modelling and Software **25**, 1014 (2010).

[5] I. Reed and X. Yu, IEEE Transactions on Acoustics, Speech, and Signal Processing **38**, 1760 (1990).

[6] Jet Propulsion Lab, California Institute of Technology, NASA, Aviris, 2022.

[7] Jet Propulsion Lab, California Institute of Technology, NASA, Aviris Science and Applications, Snow and Ice, 2022.

[8] D. M. Cvetković, M. Doob, and H. Sachs, *Spectra of Graphs: Theory and Applications* (Johann Ambrosius Barth, Leipzieg, Germany, 1995).

[9] F. R. K. Chung, *Spectral Graph Theory*, No. no. 92 in *CBMS Regional Conference Series* (Conference Board of the Mathematical Sciences, Poughkeepsie, New York, 1997).

[10] R. Bapat, A. Lal, and S. Pati, Linear Algebra and its Applications **416**, 799 (2006).

[11] K. H. Zou *et al.*, Academic Radiology **11**, 178 (2004).

[12] C.-I. Chang and S.-S. Chiang, IEEE Transactions on Geoscience and Remote Sensing **40**, 1314 (2002).

[13] J. R. P. Leo J. Grady, *Discrete Calculus: Applied Analysis on Graphs for Computational Science*, illustrated ed. (Springer Sciences and Business Media, Berlin, Germany, 2010).