UNLV University Libraries
University of Nevada, Las Vegas

UNLV Theses, Dissertations, Professional Papers, and Capstones

May 2019

Classification of Vegetation in Aerial Imagery via Neural Network

Gevand Balayan
gevand@outlook.com

Follow this and additional works at: https://digitalscholarship.unlv.edu/thesesdissertations

 Part of the Artificial Intelligence and Robotics Commons, and the Computer Engineering Commons

Repository Citation

Balayan, Gevand, "Classification of Vegetation in Aerial Imagery via Neural Network" (2019). *UNLV Theses, Dissertations, Professional Papers, and Capstones*. 3568.
https://digitalscholarship.unlv.edu/thesesdissertations/3568

This Thesis is protected by copyright and/or related rights. It has been brought to you by Digital Scholarship@UNLV with permission from the rights-holder(s). You are free to use this Thesis in any way that is permitted by the copyright and related rights legislation that applies to your use. For other uses you need to obtain permission from the rights-holder(s) directly, unless additional rights are indicated by a Creative Commons license in the record and/or on the work itself.

This Thesis has been accepted for inclusion in UNLV Theses, Dissertations, Professional Papers, and Capstones by an authorized administrator of Digital Scholarship@UNLV. For more information, please contact digitalscholarship@unlv.edu.

CLASSIFICATION OF VEGETATION IN AERIAL IMAGERY VIA NEURAL NETWORK

By

Gevand Balayan

Bachelor of Science (B.Sc.)
University of Nevada, Las Vegas
2014

A thesis submitted in partial fulfillment
of the requirements for the

Master of Science in Computer Science

Department of Computer Science
Howard R. Hughes College of Engineering
The Graduate College

University of Nevada, Las Vegas
May 2019

© Gevand Balayan, 2019
All Rights Reserved

UNLV | GRADUATE COLLEGE

Thesis Approval

The Graduate College
The University of Nevada, Las Vegas

November 28, 2018

This thesis prepared by

Gevand Balayan

entitled

Classification of Vegetation in Aerial Imagery via Neural Network

is approved in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science
Department of Computer Science

Evangelo Yfantis, Ph.D.
Examination Committee Chair

Kathryn Hausbeck Korgan, Ph.D.
Graduate College Dean

Hal Berghel, Ph.D.
Examination Committee Member

John Minor, Ph.D.
Examination Committee Member

John Wang, Ph.D.
Graduate College Faculty Representative

Abstract

This thesis focuses on the task of trying to find a Neural Network that is best suited for identifying vegetation from aerial imagery. The goal is to find a way to quickly classify items in an image as highly likely to be vegetation(trees, grass, bushes and shrubs) and then interpolate that data and use it to mark sections of an image as vegetation. This has practical applications as well. The main motivation of this work came from the effort that our town takes in conserving water. By creating an AI that can easily recognize plants, we can better monitor the impact they make on our water resources.

Acknowledgements

I would like to thank my supervisor *Dr. Evangelos Yfantis* for guiding me through my research.

I would like to also thank the committee members for agreeing to review my thesis.

Lastly I would like to thank the GIS team at SNWA for providing me valuable data that helped my research advance.

Gevand Balayan

University of Nevada, Las Vegas

May 2019

Table of Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

1.1 Motivation

Many states have limited amount of drinking water which becomes scarcer every year due to changing climate and growth. To manage their water resources wisely they encourage their residents to replace the grass on their lawns with xeriscaping. Thus, they pay the residents to take out the grass which demands a great deal of water and replace it with gravel or desert plants demanding very little watering, or no more watering than the rainfall. The amount of money paying the residents to remove the grass and other water consuming vegetation is considerable often in the order of tens of thousands of dollars. Often, property owners take out the grass, take the money, and later on they change their mind or sell the property to another person and reseed the grass on the property. In such a case the authorities must refund money back. In order to automate the process of who maintains a lawn with grass and who does not, an unmanned aerial vehicle with LIDAR could be used to automatically recognize grass loans and vegetation areas, as well as xeriscaping for each address in a city. In this research paper we show the digital image processing and AI algorithm used on LIDAR aerial imagery in order to classify a residence as having a grass loan and vegetation, or having a certain percentage of vegetation, and the rest xeriscaping, or all xeriscaping.

1.2 Objective

As mentioned in the motivation the main objective of this thesis is to figure out an accurate and efficient way of identifying whether an aerial image contains vegetation.

In this thesis we propose a back propagation neural network whose task it is to work with a small

subsection of an input image and provide a probability of that section containing vegetation. Once each section's probability is measured the image could be reconstructed and the square footage of how much vegetation exists on the aerial image could be calculated.

The main advantages of this solution is the speed at which each section could be calculated. The backpropogation neural network is simple and requires few calculation to classify each section. The process itself could be completely parallelized since each section is independent from each other.

1.3 Outline

In the first chapter, we gave a brief overview of the problem and gave a quick summary for motivation and objective of this thesis.

In the second chapter we will introduce the background to neural networks. Then we will discuss their history, application and current research being conducted with them. We will delve deeper into backpropagation and explain the basics of how the classification works. Lastly we will discuss the limits of backpropogation and drawbacks to using them over other statistical approaches.

In the third chapter we will discuss the approach that was taken for the solution of the objective of this thesis. Starting with gathering a large enough training set, figuring out the best structure of the neural network and best values for its super parameters and the process of training the network itself.

The fourth chapter is dedicated to the actual implementation of the neural network and will discuss the results that it provided.

The fifth and final chapter is focused on talking about further improvements that could be done to this research and will have a conlusion on the research done in this paper.

Chapter 2

Background and Literature Review

2.1 Four Band Imagery

Four band aerial imagery is commonly used in GIS applications. Generally band 1, 2, and 3 are Red, Green, and Blue "true color" values. The 4th band is part of the alpha channel and is generally known as NIR (near infrared). It is commonly saved as a .tiff or .tif format. Neither tiff nor tif formats have a real differences in format between them.

The pixel value is stored as a 32 bit integer. Where the first 8 bits are the Alpha channel, the next 8 are Blue, followed by Green and finishing with Red. The below code sample shows how to extract a band value of pixel given an input integer by using the logical and operator with 1111 1111 8 bit mask and shifting properly.

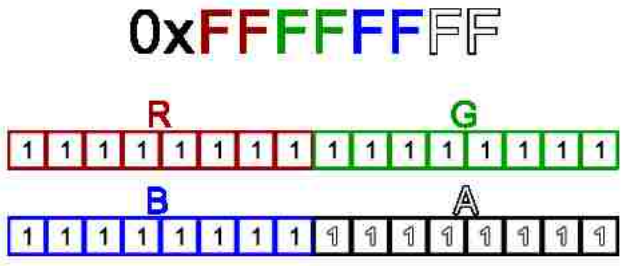

Figure 2.1: Structure of a pixel

```
public static int GetR(int pixel)
  return (pixel & 0xff);
public static int GetG(int pixel)
  return ((pixel >> 8) & 0xff);
```

```
public static int GetB(int pixel)
  return ((pixel >> 16) & 0xff);
public static int GetA(int pixel)
  return ((pixel >> 24) & 0xff);
```

2.2 Backpropagation Neural Networks

Back Propagation (BP) refers to a broad family of Artificial Neural Networks (ANN), whose architecture consists of different interconnected layers. [1] The BP ANNs represents a kind of ANN, whose learning algorithm is based on the Deepest-Descent technique. If provided with an appropriate number of Hidden units, they will also be able to minimize the error of nonlinear functions of high complexity. [1]

A neural network consists of neurons. They are non-linear processing elements that will sum the incoming signals in order to generate an output signal via a pre-defined non-linear function. The neurons are connected by terms with variable weights. The output of one neuron multiplied by a weight becomes the input of an adjacent neuron of the next layer. [2]

The neurons are arranged into sections called layers. A neural network will always have an input layer and an output layer. In between the two layers there could be any number of middle layers known as hidden layers.

The goal of training a neural network is to find weights that will result the minimization of the error signal in the output layer. The most common form of machine learning is supervised learning. Supervised learning consists of presenting an input vector into the network's input layer. The network's output is calculated, and compared to the true value of the input layer. The error between the two is then used in the adjustment of weights until the error reaches acceptable levels.

2.3 Neural Network Hyper Parameters

The equation below shows how to use steepest descent in order to calculate the network parameters θ at step t. With back propagation, the goal is to use the previous θ at step $t-1$ and subtract the minimization of loss or error function L with respect to θ time the learning rate ϵ_t. A mini-batch size B could be used to have multiple input vectors be part of an iteration.

$$\theta^{(t)} \leftarrow \theta^{(t-1)} - \epsilon_t \frac{1}{B} \sum_{t'=Bt+1}^{B(t+1)} \frac{\partial L(z_{t'}, \theta)}{\partial \theta}$$

With batch size of 1, the equation simple becomes:

$$\theta^{(t)} \leftarrow \theta^{(t-1)} - \epsilon_t \frac{\partial L(z_{t'}, \theta)}{\partial \theta}$$

Thus the hyper parameters of a ANN is the loss/error function L usually represented as

$$L = \frac{1}{2}(y_i - z_i)^2$$

where y_i is the true value of the input and z_i is the output generated by the ANN. The learning rate ϵ_t governs the rate at which the model tries to converge, the mini-batch size measures the number of input vectors used per iterations and the set T denotes the number of training set iterators used in supervised learning.

2.4 Other methods of classification

An alternative to ANN are the Random Forest algorithms. They are both classification algorithms and use supervised learning. A random forest is made out of many decision trees. Given an input data the decision trees all evaluate and classify the data. After that the classification which occurred the most wins out and is chosen by the forest as the correct one.

2.5 Literature Review

Artificial intelligence is a hot topic in the field of computer science. Technological leaders such as Google, Amazon, Microsoft and Facebook are all investing heavily in building AIs that will give them advantage in the market. Since the focus of this paper is on research done in classifying aerial imagery, I chose to focus my research on topics similar to that nature.

2.5.1 Vehicle Detection in Aerial Imagery

Vehicle detection with orientation estimation in aerial images has received widespread interest as it is important for intelligent traffic management. This is a challenging task, not only because of the complex background and relatively small size of the target, but also the various orientations of vehicles in aerial images captured from the top view.[3] The researchers chose to use a a feed-forward convolutional neural network (CNN) named Oriented_SSD (Single Shot MultiBox Detector, SSD). [3]

The group chose to base their research on the VGG-16 convolution network proposed by Simonyan, K., Zisserman, A. in 2014. [4] Figure 2.2 shows the structure of the neural network's

convolution. A 512 by 512 image goes through 5 VGG16 layers, after which additional detection layers are added as layer 6 through 11. Figure 2.3 explains the architecture of the VGG16 lair.

The main difference between the work for aerial imagery detection is the method used in classification of the images. With vegetation having the benefit of multi-band imagery allows to use a simpler form of a back propogation neural network, however since the researchers were only working with basic three band images, a convolutional layer is needed for further accuracy in classification.[5]

A future research topic could be on combining the use of convolutional neural networks proposed with the multi-band imagery in order to see if the accuracy of the results improves.

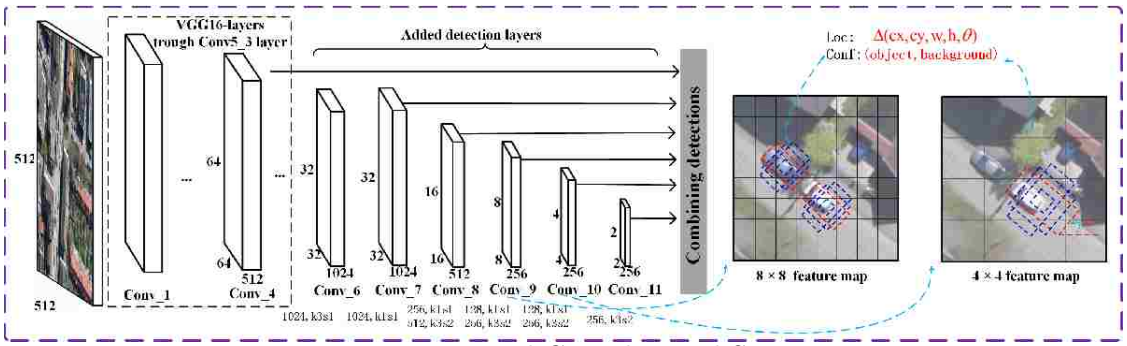

Figure 2.2: Proposed Convolutional Structure

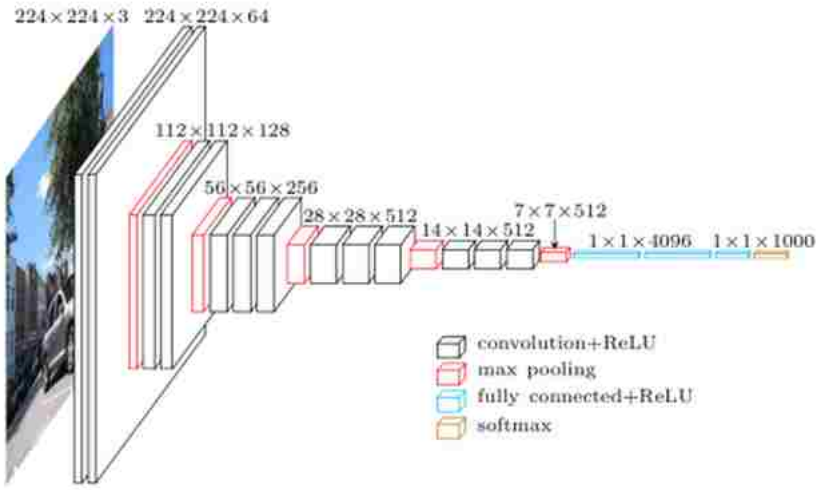

Figure 2.3: VGG 16

Chapter 3

Method

The main objective of this thesis and the problem statement is to *figure out an accurate and efficient way of identifying whether an aerial image contains vegetation using a back propagation neural network*. This chapter will go over the methodologies used to solve this problem.

3.1 Gathering a Training Set

In order to create a back propagation neural network a training set is needed in order to allow the neural network to make a proper classification. In order to achieve a high degree of accuracy said test must be robust in size, diverse in its contents and simple to consume in training. The first step in the process was to find aerial imagery that already had classified whether an object in it was vegetation or not.

I turned to my colleagues at SNWA who collected 4 band 3 inch aerial imagery. Each image file was roughly .5 to .75 GB in size, covering an area of 100s of square feet. The image needed to be split up into numerous smaller sections (1000s per single photo). Each sub image would then be classified using the 4th band of the image.

Since the 4th band would be used to train the network, the 4 band values could be mathematically transformed to have high infrared values stand out. The figures below show the differences between true color, basic infrared and sharpened infrared image section.

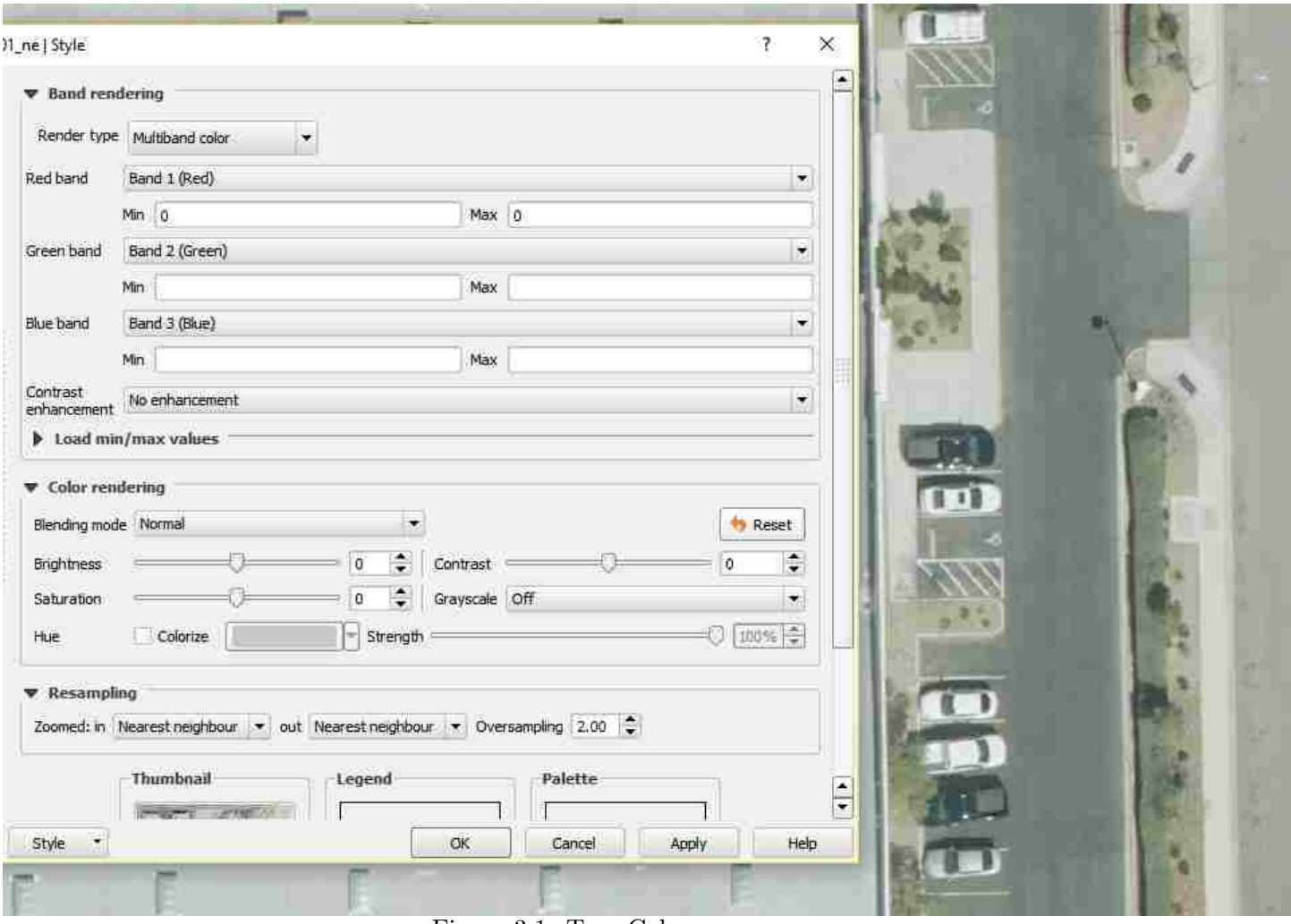

Figure 3.1: True Color

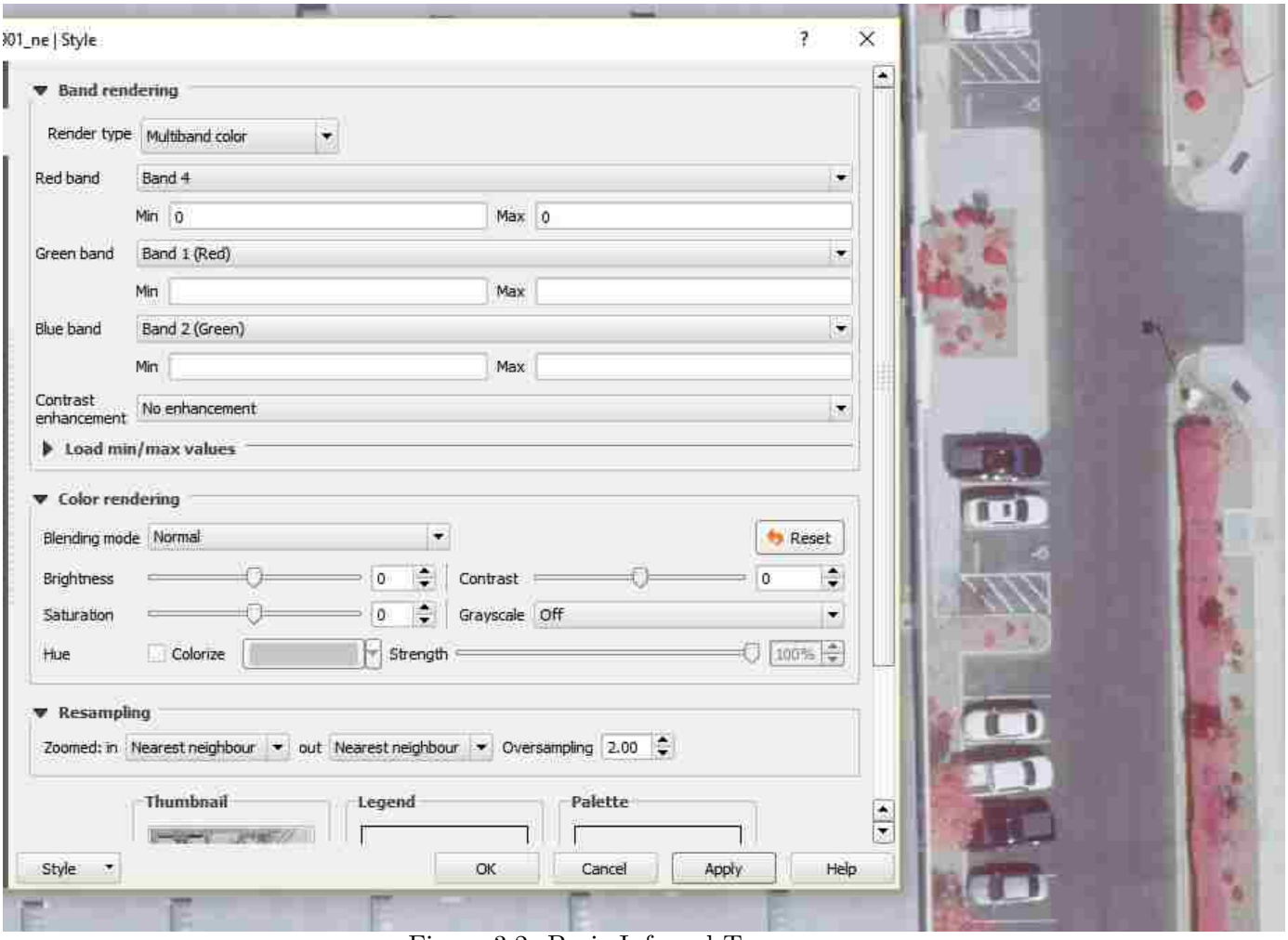

Figure 3.2: Basic Infrared T_n

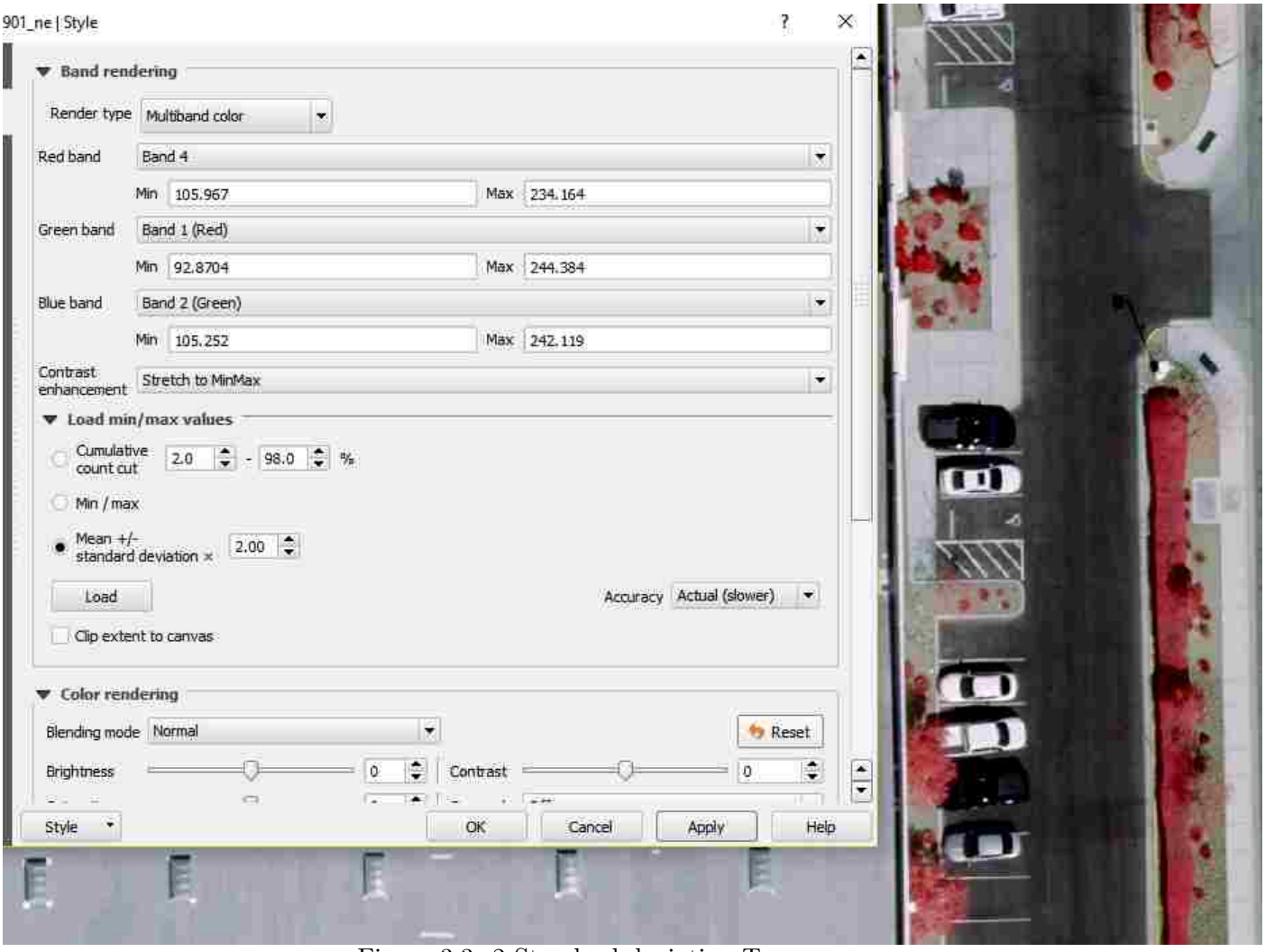

Figure 3.3: 2 Standard deviation T_d

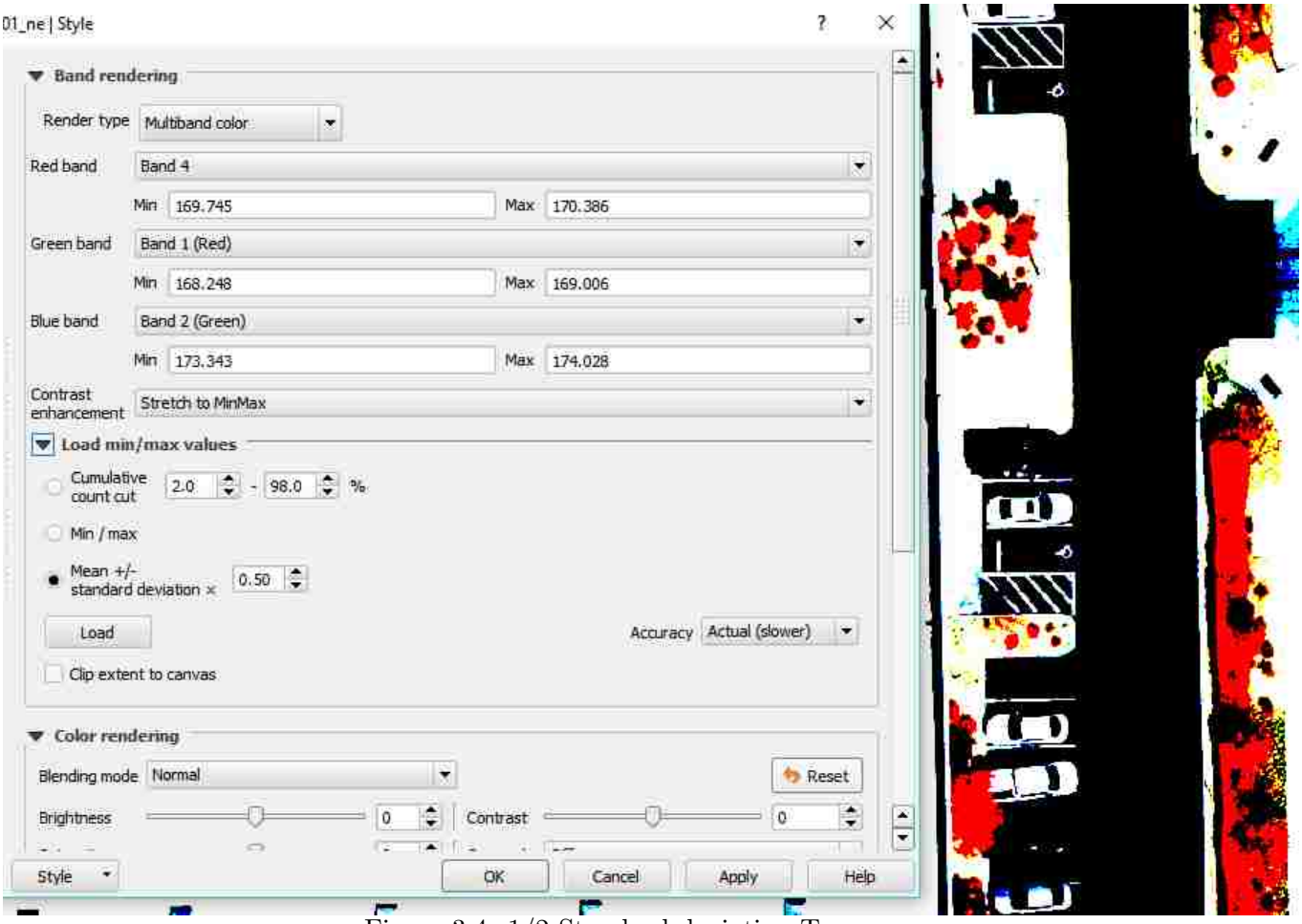

Figure 3.4: 1/2 Standard deviation T_h

Each image was split into 10x10 squares, each in need of identification of whether or not its contains grass or not. The training sets were generated for every type of imagery. Covering basic infrared, 2 standard deviation of infrared, and for 1/2 standard deviation of infrared.

The classification of the training set image varied on two parameters. One being how many of the 100 pixels were counted as vegetation, second being what Threshold value of the infrared channel would classify the pixel as vegetation. Each set had a total of 6 classifications shown below.

Classifcation	Number Of Vegetation Pixels
0 0 0 0 0 1	0
0 0 0 0 1 0	1 to 20
0 0 0 1 0 0	21 to 40
0 0 1 0 0 0	41 to 60
0 1 0 0 0 0	61 to 80
1 0 0 0 0 0	81 to 100

Table 3.1: Training Classification

Each training set competed in the training of neural networks. The motivation was to figure which training set will perform best with different neural network structures and super parameters. Since the aerial imagery is updated every year, it would be important to retrain the neural network to adjust to the possibility of a new camera or new resolution. Thus it is important to know what infrared values are best used in order to classify a test set properly.

A single test set was created, spanning across all types of infrared in order to validate which training set worked best. An item would only be added to the test set if it was classified the same by each training set.

3.2 Result of Generating Training Sets

The table below shows the training sets generated from the three types of infrared images. Each member of the vegetation set size, is one 10x10 image that has atleast 1 pixel that is marked by the 4th band as vegetation. Non Vegetation set contains all images that do not have a single pixel marked as vegetation by the 4th band.

In order to have a single test set, 7,200 10x10 sections that were classified the same in all three training sets were removed out of them and added to a single test set below. The figure below illustrates the relationship between the sets.

Name	Deviation	Vegetation Set Size	Non Vegetation Set Size
T_n	None	9,333	114,764
T_d	2	11,271	113,826
T_h	.5	14,122	109,975

Table 3.2: Training sets

Name	Vegetation Set Size	Non Vegetation Set Size
T_t	3,208	3,992

Table 3.3: Test set

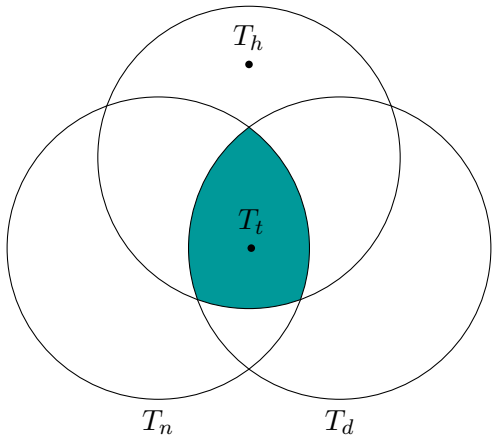

Figure 3.5: Relationship between the four sets.

3.3 Neural Network Structure

The neural network structure used in the project was a simple 3 layer back propagation model. The input layer would vary in two ways, one would contain 3 nodes per pixle of the section, the other would contain only one. Each node would either signify the R, G, B values of the pixel or, in case of a single neuron per input layer, would signify the fourth channel. Thus if the section is 10x10 square, containing 100 pixels, a total of 300 input layer nodes would be used in neural network 1, N_{rgb} and 100 input neurons for neural network two, N_i The number of hidden layer neurons would vary, when trying to minimize the error. The output layer would always contain 6 neurons as mentioned above and would result in a vector $[abcdef]$ with all values between 0 and 1. The maximum of the vector would be treated as 1 and the rest of the values transformed to 0. Figure 3.1 illustrates the basic structure of the network used below.

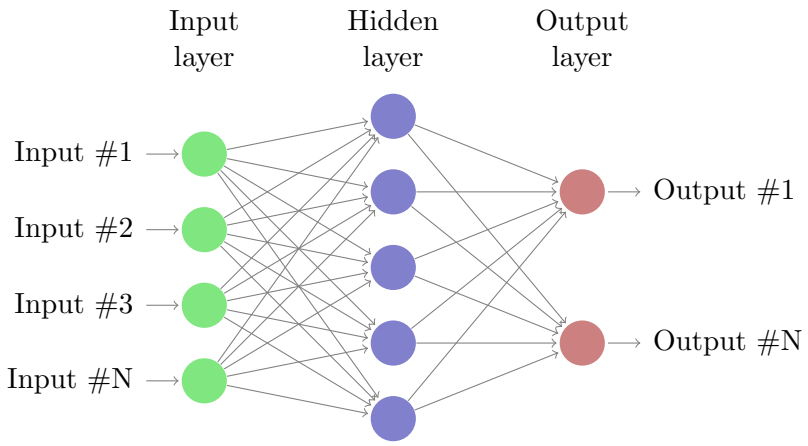

Figure 3.6: Typical example of a neural network.

The reason for having two Neural Networks was to be able to identify vegetation with two types of input. One first input type being the 4th band of the images, the other was the true color RGB values. N_{rgb}, would be useful in situations where an image does not have a 4th band. With the three training sets T_n, T_d and T_h, it is now important to identify which hyper parameters would serve best for classification of vegetation of aerial imagery.

3.4 Choosing Hyper Parameters

The goal of selecting the proper hyper parameters is to come up with a neural network which gives the smallest error over classifying the test set T_t. However since there were 3 training sets T_n, T_d and T_h, it is important to train the constructed neural network independently with all training sets and then compare the resulting errors.

As mentioned before, the size of each training image is 10x10, which resulted in 100 pixels and required 300 input neurons for N_{rgb} and 100 inputs for N_i. After that the number of hidden layers was varied between 5, 30 and 100. The activation function for each neuron would be either the Sigmoid, hyperbolic tangent or the ReLU functions. The rate of change $epsilon_t$ would vary between 0.001 and 0.01. The batch size would remain at 1. The loss function used was the sum of square error.

The initial value of the learning rate was 0.1, and the process was online learning meaning that every sample from the learning set as it went through the neural network, starting from the input layer, going through the hidden layer and then through the output layer, propagated the error backwards adjusting the weights of the output layer and the hidden layer so that the total

error was reduced in every trial. If for one sample going through the total error increased then the learning rate was reduced and the process restarted.[6]

The training first started with 200 samples for each classification for a total of 1200. 40 epochs were used to further adjust the weights. The reason 40 epochs were used is that the error would generally stop improving after 40 and would oscillate between two values. Future trials would see an increase of samples to 400 per each classification all the way up to 2,400 samples selected from every classification for a total of 14,400 samples.[6]

There were a total of 50 permutations of neural networks that were tested using these hyper parameters, the table below shows results of the most successful hyper parameters for every hidden layer count. The $epsilon_t$ (learning rate) that allowed for the best convergence was 0.001. While other $epsilon_t$ were tested they were not able to converge as quickly as 0.001.

The figure below shows some of the performance for both networks, which training set was used and which hyper parameters were shown. Accuracy is the percent of the testing set that was classified correctly. The last two parameters are the networks that performed best and were used.

NN Type	Hidden Layer	$epsilon_t$	Test Set Size	F(x)	T_n accuracy	T_d accuracy	T_h accuracy
N_{rgb}	5	.01	1,200	RelU	81	83	78
N_{rgb}	5	.01	1,200	Sigmoid	80	83	79
N_{rgb}	5	.01	1,200	Tanh	81	83	78
N_{rgb}	30	.005	2,400	RelU	84	83	80
N_{rgb}	30	.005	2,400	Sigmoid	89	91	83
N_{rgb}	30	.005	2,400	Tanh	88	95	82
N_{rgb}	30	.005	6,000	RelU	89	93	98
N_{rgb}	30	.005	6,000	Sigmoid	91	90	90
N_{rgb}	30	.005	6,000	Tanh	90	92	88
N_{rgb}	100	.001	14,400	RelU	95	96	91
N_{rgb}	100	.001	14,400	Sigmoid	94	91	92
N_{rgb}	100	.001	14,400	Tanh	91	92	90
N_i	5	.01	1,200	RelU	74	71	78
N_i	5	.01	1,200	Sigmoid	71	76	70
N_i	5	.01	1,200	Tanh	77	78	70
N_i	30	.005	2,400	RelU	80	80	81
N_i	30	.005	2,400	Sigmoid	81	83	84
N_i	30	.005	2,400	Tanh	88	89	80
N_i	30	.005	6,000	RelU	81	82	78
N_i	30	.005	6,000	Sigmoid	84	84	82
N_i	30	.005	6,000	Tanh	81	83	82
N_i	100	.001	14,400	RelU	83	83	81
N_i	100	.001	14,400	Sigmoid	88	84	81
N_i	100	.001	14,400	Tanh	85	86	88
N_i	**30**	**.001**	**14,400**	**Tanh**	**94**	**98**	**96**
N_{rgb}	**100**	**.001**	**14,400**	**Tanh**	**91**	**92**	**91**

Table 3.4: Hyper parameters performance

3.5 Training Conclusion

The set that generally performed the best was the set T_d. It shows that having a smaller variance between the 4th band values in the image helps the selection of higher quality test samples, however, if the variance is too small, like in the set T_h, the test set does not perform well. Thus the set T_h seemed to have performed the worst. The activation F(x) Tanh performed best, slightly edging out RelU, and the N_i neural network was able to reach 98 percent accuracy in identifying images using the 4th infrared band, while the set N_{rgb} could only get to 92 percent accuracy. If the 4th band is available, it would seem that simply using the infrared as input and ignoring the RGB values is the optimal way to go.

Chapter 4

Applying the Neural Networks in Practice

To apply the N_{rgb} neural network in practice a small program was implemented that would take in an image, separate it into 10x10 section and then feed each section as the input into the neural network. If the section was classified as vegetation by the network, it was colored purple, otherwise it was left as is.

4.1 Implementation Tools

The implementation of the program relied on tools. The usage of these tools will be discussed in this section.

4.1.1 Hardware and Operating System

All research was done on my personal computer. Below are its specifications:

Operating System	Windows 10 Pro, 64 Bit
Processor	Intel(R) Pentium(R) CPU G3258 @ 3.20GHz
RAM	8 GB
Graphics Card	AMD Radeon R9 200 Series
Hard Disc	1TB

Table 4.1: Specifications

4.1.2 Programming Language and IDE

All work for this project was done in Microsoft's Visual Studio IDE by using the C# programming language. C# allowed me to quickly build a windows form application that would bring the neural network to practical use.

4.1.3 Neural Network Library

The C# Encog library was used to help with implementation of the neural networks. The library is licensed under Apache License 2.0 and is has its source open for browsing on github. [1]. It was used mainly for the purpose of handling Input/Output of image data as it has a robust library that deals well with all types of image data. A custom NeuralNetwork class and BackPropogation trainer was written for the purposes of this paper.

4.1.4 Parallelization

Since each section of the image could be computed in parallel, majority of the program could be considered perfectly parallel. Meaning a significant speed up will occur as more resources are added to the machine.

4.2 Results

The neural network would create a vector $[abcdef]$. The program started by marking all sections as vegetation the vector indicated a result of between .9 and 1, showing all 10x10 section that would have atleast 90 pixels classified as vegetation. After that it was expanded to mark all vegetation that would have at least half its pixels classified as vegetation. Lastly the program simply marks all sections have a non zero amount of vegetation. The below pictures show the results.

[1] https://github.com/encog/encog-dotnet-core

Figure 4.1: True color image

Figure 4.2: Between .9 and 1

Figure 4.3: Between .5 and 1

Figure 4.4: Not 0

Chapter 5

Future Improvements and Possible Research Topics

The neural network proposed in this thesis performed quite well however there could be future improvements that could be part of a different research paper. The first part that could improve the solution is having a better training set.

5.1 Better Training Set

Creating the training set required using a 4 band image where the first three bands were the standard R, G, B bands with the 4th being an infrared band. The most advanced imagery can go up to 8 bands, with Thermal, Shortwave/Longwave Infrared, Panchromatic, Cirrus bands. This type of imagery, while hard to obtain, would provide a much more accurate training set for the neural network and would allow higher accuracy in vegetation analysis.

5.2 Multiple Neural Networks

Another way to improve the accuracy of the solution is to research the impact of having multiple neural networks working together on the same image. A neural network that is specifically training to identify trees could complement a neural network that targets grass, while another neural network that is trained to identify shrubs and bushes is focusing on its task. By having multiple neural networks trained focused in on specific tasks, it may be possible to significantly improve the results of identifying imagery.

5.3 Conclusion

In conclusion, the research was able to successfully find a method of classifying vegetation in aerial imagery. Using the multi-band aerial imagery it was possible to extract an accurate training set. Using that training set numerous neural network structures were tested with different hyper parameters and the most accurate one was selected. After that the neural network as used part of a testing application that could be used to visually mark any vegetation provided aerial imagery.

Appendix A

Source Code

```
1
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Text;
6  using System.Threading.Tasks;
7  using System.Drawing;
8  using System.IO;
9  using BitMiracle.LibTiff.Classic;
10 using System.Drawing.Imaging;
11
12 namespace Neural.Utilities
13 {
14     public class TiffConverter
15     {
16         public static int Split(string tiffFilePath, int splitPixelSize
               = 100)
17         {
18             string fileName = Path.GetFileName(tiffFilePath).Replace(".
                   tif", "");
19             using (Tiff image = Tiff.Open(tiffFilePath, "r"))
20             {
21                 // Find the width and height of the image
```

```
22                  FieldValue[] value = image.GetField(TiffTag.IMAGEWIDTH)
                        ;
23                  int width = value[0].ToInt();
24
25                  value = image.GetField(TiffTag.IMAGELENGTH);
26                  int height = value[0].ToInt();
27
28                  int imageSize = height * width;
29                  int[] raster = new int[imageSize];
30                  image.ReadRGBAImage(width, height, raster);
31                  int total = imageSize / splitPixelSize;
32                  int rowCount = 0;
33                  int colCount = 0;
34                  // Read the image into the memory width
35                  while (2000 >= rowCount * colCount)
36                  {
37                      try
38                      {
39
40                          int counter = 0;
41                          using (Bitmap bmp = new Bitmap(splitPixelSize,
                                splitPixelSize))
42                          {
43
44                              for (int i = 0; i < bmp.Width; ++i)
45                                  for (int j = 0; j < bmp.Height; ++j)
46                                  {
47                                      int x = i + (colCount *
                                            splitPixelSize);
48                                      int y = j + (rowCount *
                                            splitPixelSize);
49
50                                      int offset = (height - y - 1) *
                                            width + x;
51                                      int red = Tiff.GetA(raster[offset])
                                            ;
```

```
52                                  int green = Tiff.GetR(raster[offset
                                        ]);
53                                  int blue = Tiff.GetG(raster[offset
                                        ]);
54                                  if ((red > 160 && red < 200) && (
                                        green > 80 && green < 120) && (
                                        blue > 80 && blue < 120))
55                                  {
56                                      counter++;
57                                  }
58
59                                  bmp.SetPixel(i, j, Color.FromArgb(
                                        red, green, blue));
60                              }
61                          }
62                          using (Bitmap bmp = new Bitmap(splitPixelSize,
                                splitPixelSize))
63                          {
64                              for (int i = 0; i < bmp.Width; ++i)
65                                  for (int j = 0; j < bmp.Height; ++j)
66                                  {
67                                      int x = i + (colCount *
                                            splitPixelSize);
68                                      int y = j + (rowCount *
                                            splitPixelSize);
69
70                                      int offset = (height - y - 1) *
                                            width + x;
71                                      int red = Tiff.GetR(raster[offset])
                                            ;
72                                      int green = Tiff.GetG(raster[offset
                                            ]);
73                                      int blue = Tiff.GetB(raster[offset
                                            ]);
74                                      bmp.SetPixel(i, j, Color.FromArgb(
                                            red, green, blue));
```

```
75                                      }
76                                  string path = @"D:\Imagery\_Nothing";
77                                  if (counter > 5)
78                                      path = @"D:\Imagery\_Vege";
79                                  else if (counter == 0)
80                                      path = @"D:\Imagery\_Nothing";
81                                  else
82                                      continue;
83                                  path = System.IO.Path.Combine(path, String.
                                        Format("{0}{1}x{2}_real.bmp", fileName,
                                        rowCount, colCount));
84                                  bmp.Save(path);
85                              }
86
87                          }
88                          catch { }
89                          finally
90                          {
91                              colCount++;
92                              if (colCount % 10 == 0)
93                              {
94                                  colCount = 0;
95                                  rowCount++;
96                              }
97                          }
98                      }
99                      return rowCount;
100                 }
101             }
102         }
103 }
104 using Microsoft.VisualStudio.TestTools.UnitTesting;
105 using Neural.Utilities;
106 using System;
107 using System.Collections.Generic;
108 using System.IO;
```

```
109 using System.Linq;
110 using System.Text;
111 using System.Threading.Tasks;
112
113 namespace NeuralNet.Test
114 {
115     [TestClass]
116     public class TiffConverterTest
117     {
118         [TestMethod]
119         public void TiffConverterTest_Split()
120         {
121             string[] fileEntries = Directory.GetFiles(@"D:\Imagery\");
122             int counter = 0;
123             foreach (var file in fileEntries)
124             {
125                 if (file.Contains(".tif"))
126                 {
127                     counter++;
128                     if (counter > 100)
129                         break;
130                     var byteRay = TiffConverter.Split(file, 10);
131                 }
132             }
133
134         }
135     }
136 }
137 namespace GrassIdentifier
138 {
139     partial class Form1
140     {
141         /// <summary>
142         /// Required designer variable.
143         /// </summary>
144         private System.ComponentModel.IContainer components = null;
```

```
145
146         /// <summary>
147         /// Clean up any resources being used.
148         /// </summary>
149         /// <param name="disposing">true if managed resources should be
                disposed; otherwise, false.</param>
150         protected override void Dispose(bool disposing)
151         {
152             if (disposing && (components != null))
153             {
154                 components.Dispose();
155             }
156             base.Dispose(disposing);
157         }
158
159         #region Windows Form Designer generated code
160
161         /// <summary>
162         /// Required method for Designer support - do not modify
163         /// the contents of this method with the code editor.
164         /// </summary>
165         private void InitializeComponent()
166         {
167             this.button1 = new System.Windows.Forms.Button();
168             this.SuspendLayout();
169             //
170             // button1
171             //
172             this.button1.Location = new System.Drawing.Point(33, 25);
173             this.button1.Name = "button1";
174             this.button1.Size = new System.Drawing.Size(135, 51);
175             this.button1.TabIndex = 0;
176             this.button1.Text = "OpenFile";
177             this.button1.UseVisualStyleBackColor = true;
178             this.button1.Click += new System.EventHandler(this.
                    button1_Click);
```

```
179             //
180             // Form1
181             //
182             this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F
                    );
183             this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.
                    Font;
184             this.ClientSize = new System.Drawing.Size(215, 100);
185             this.Controls.Add(this.button1);
186             this.Name = "Form1";
187             this.Text = "Form1";
188             this.ResumeLayout(false);
189
190         }
191
192         #endregion
193
194         private System.Windows.Forms.Button button1;
195     }
196 }
197 using System;
198 using System.Collections.Generic;
199 using System.ComponentModel;
200 using System.Data;
201 using System.Drawing;
202 using System.Linq;
203 using System.Text;
204 using System.Threading.Tasks;
205 using System.Windows.Forms;
206
207 namespace GrassIdentifier
208 {
209     public partial class NeuralForm : Form
210     {
211         public NeuralForm()
212         {
```

```
213             InitializeComponent();
214         }
215 
216         public Bitmap BitMap { get; internal set; }
217 
218         private void NeuralForm_Load(object sender, EventArgs e)
219         {
220 
221             this.Width = BitMap.Width;
222             this.Height = BitMap.Height;
223             pbOutPut.Image = BitMap;
224         }
225     }
226 }
227 namespace GrassIdentifier
228 {
229     partial class NeuralForm
230     {
231         /// <summary>
232         /// Required designer variable.
233         /// </summary>
234         private System.ComponentModel.IContainer components = null;
235 
236         /// <summary>
237         /// Clean up any resources being used.
238         /// </summary>
239         /// <param name="disposing">true if managed resources should be
                disposed; otherwise, false.</param>
240         protected override void Dispose(bool disposing)
241         {
242             if (disposing && (components != null))
243             {
244                 components.Dispose();
245             }
246             base.Dispose(disposing);
247         }
```

```
248
249         #region Windows Form Designer generated code
250
251         /// <summary>
252         /// Required method for Designer support - do not modify
253         /// the contents of this method with the code editor.
254         /// </summary>
255         private void InitializeComponent()
256         {
257             this.pbOutPut = new System.Windows.Forms.PictureBox();
258             ((System.ComponentModel.ISupportInitialize)(this.pbOutPut))
                    .BeginInit();
259             this.SuspendLayout();
260             //
261             // pbOutPut
262             //
263             this.pbOutPut.Location = new System.Drawing.Point(12, 12);
264             this.pbOutPut.Name = "pbOutPut";
265             this.pbOutPut.Size = new System.Drawing.Size(172, 69);
266             this.pbOutPut.SizeMode = System.Windows.Forms.
                    PictureBoxSizeMode.AutoSize;
267             this.pbOutPut.TabIndex = 0;
268             this.pbOutPut.TabStop = false;
269             //
270             // NeuralForm
271             //
272             this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F
                    );
273             this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.
                    Font;
274             this.ClientSize = new System.Drawing.Size(192, 89);
275             this.Controls.Add(this.pbOutPut);
276             this.Name = "NeuralForm";
277             this.Text = "NeuralForm";
278             this.Load += new System.EventHandler(this.NeuralForm_Load);
```

```
279             ((System.ComponentModel.ISupportInitialize)(this.pbOutPut))
                    .EndInit();
280             this.ResumeLayout(false);
281             this.PerformLayout();
282 
283         }
284 
285         #endregion
286 
287         private System.Windows.Forms.PictureBox pbOutPut;
288     }
289 }
290 
291 namespace GrassIdentifier
292 {
293     public partial class Form1 : Form
294     {
295         public Form1()
296         {
297             InitializeComponent();
298         }
299 
300         public const string NETWORK_PATH = @"Resources\network_10.
                parameters";
301         public const int SIZE = 10;
302         private void button1_Click(object sender, EventArgs e)
303         {
304             using (OpenFileDialog dlg = new OpenFileDialog())
305             {
306                 dlg.Title = "Open Image";
307                 dlg.Filter = "Image files (*.bmp, *.jpg, *.jpeg, *.jpe,
                        *.jfif, *.png) | *.bmp; *.jpg; *.jpeg; *.jpe; *.
                    jfif; *.png";
308 
309                 if (dlg.ShowDialog() == DialogResult.OK)
310                 {
```

```
311                     var bitMap = new Bitmap(dlg.FileName);
312                     NeuralForm n = new NeuralForm();
313                     n.BitMap = bitMap;
314                     n.Show();
315
316                     //
317                     var network = new NeuralNetwork();
318                     FileInfo networkFile = new FileInfo(NETWORK_PATH);
319                     if (System.IO.File.Exists(NETWORK_PATH))
320                         network = (NeuralNetwork)(NeuralNetwork(
                                networkFile));
321
322
323                     int hCounter = 0; int Wcounter = 0;
324                     int hMax = bitMap.Height / SIZE;
325                     int wMax = bitMap.Width / SIZE;
326                     var outputBitmap = new Bitmap(bitMap.Width, bitMap.
                        Height);
327                     while (hMax > hCounter)
328                     {
329                         Encog.ML.Data.Image.ImageMLDataSet testingSet =
                                new Encog.ML.Data.Image.ImageMLDataSet(new
                                Encog.Util.DownSample.RGBDownsample(), false
                                , 1, -1);
330                         var _SizexSizebitmap = new Bitmap(SIZE, SIZE);
331
332                         for (int i = 0; i < SIZE; i++)
333                         {
334
335                             for (int j = 0; j < SIZE; j++)
336                             {
337                                 var px = bitMap.GetPixel(Wcounter *
                                        SIZE + j, hCounter * SIZE + i);
338                                 _SizexSizebitmap.SetPixel(i, j, px);
339                             }
340                         }
```

```
341
342                     ImageMLData data = new ImageMLData(
                            _SizexSizebitmap);
343                     testingSet.Add(data, null);
344                     testingSet.Downsample(SIZE, SIZE);
345                     bool isVege = false;
346
347                     foreach (IMLDataPair pair in testingSet)
348                     {
349                         IMLData output = network.Compute(pair.Input
                                );
350                         if (output[5] < output[0] ||
351                 output[5] < output[1] ||
352                 output[5] < output[2] ||
353                 output[5] < output[3] ||
354                 output[5] < output[4] ||)
355                             isVege = true;
356                     }
357                     for (int i = 0; i < SIZE; i++)
358                     {
359
360                         for (int j = 0; j < SIZE; j++)
361                         {
362
363                             var px = _SizexSizebitmap.GetPixel(i, j
                                    );
364                             if (!isVege)
365                                 outputBitmap.SetPixel( Wcounter *
                                        SIZE + j, hCounter * SIZE + i,
                                        px);
366                             else
367                                 outputBitmap.SetPixel( Wcounter *
                                        SIZE + j, hCounter * SIZE + i,
                                        Color.FromArgb(px.R , 0, px.B));
368                         }
369                     }
```

```
370
371                             Wcounter++;
372                             if (wMax <= Wcounter)
373                             {
374                                 Wcounter = 0;
375                                 hCounter++;
376                             }
377
378                         }
379
380
381                         NeuralForm n2 = new NeuralForm();
382                         n2.BitMap = outputBitmap;
383                         n2.Show();
384
385
386                     }
387                 }
388             }
389         }
390 }
```

Bibliography

[1] Dr. Massimo Buscema. Back Propagation Neural Networks. *Substance Use & Misuse*, 33:233–265, 1998.

[2] Dr. Fu-Chuang Chen. Back-Propagation Neural Networks for Nonlinear Self-Tuning Adaptive Control . *IEEE Control Systems Magazine*, 10:44 – 48, 1990.

[3] Tianyu Tang; Shilin Zhou; Zhipeng Deng Lin Lei and Huanxin Zou. Arbitrary-Oriented Vehicle Detection in Aerial Imagery with Single Convolutional Neural Networks. *ISPRS Journal of Photogrammetry and Remote Sensing*, 9, 2017.

[4] A. Simonyan, K.; Zisserman. Very deep convolutional networks for Large-Scale image recognition. 2014.

[5] Harris SL Yfantis EA. An Autonomous UAS with AI for Forest Fire Preven, Detection. *J Comp Sci Appl Inform Tehcnol*, pages 2 – 5, 2017.

[6] Gevand Balayan Yfantis EA. Classification of Vegetation Areas Using LIDAR Images and AI. *Symbiosis*, pages 1–6, 3(3).

Curriculum Vitae

Graduate College
University of Nevada, Las Vegas

Gevand G. Balayan
gevand@outlook.com

Degrees:

Bachelor of Science in Computer Science 2014
University of Nevada Las Vegas

Thesis Title: Classification of Vegetation in Aerial Imagery via Neural Network

Thesis Examination Committee:

Chairperson, Dr. E. A. Yfantis, Ph.D.
Committee Member, Dr. H. Berghel, Ph.D.
Committee Member, Dr. John T. Minor, Ph.D.
Graduate Faculty Representative, Dr. John Wang, Ph.D.