Pixel Based Image Classification

Goal

- Classify pixels in different 50 photographs that include the same classes.

Overview

- Sample Training, Testing, and Validation Set
- Pre-processed imagery
    - Color Space Transformations
    - Image Segmentation (Smoothing)
    - Texture Calculations (GLCM)
- Train random forest model
- Tune Model
- Classify all 50 images

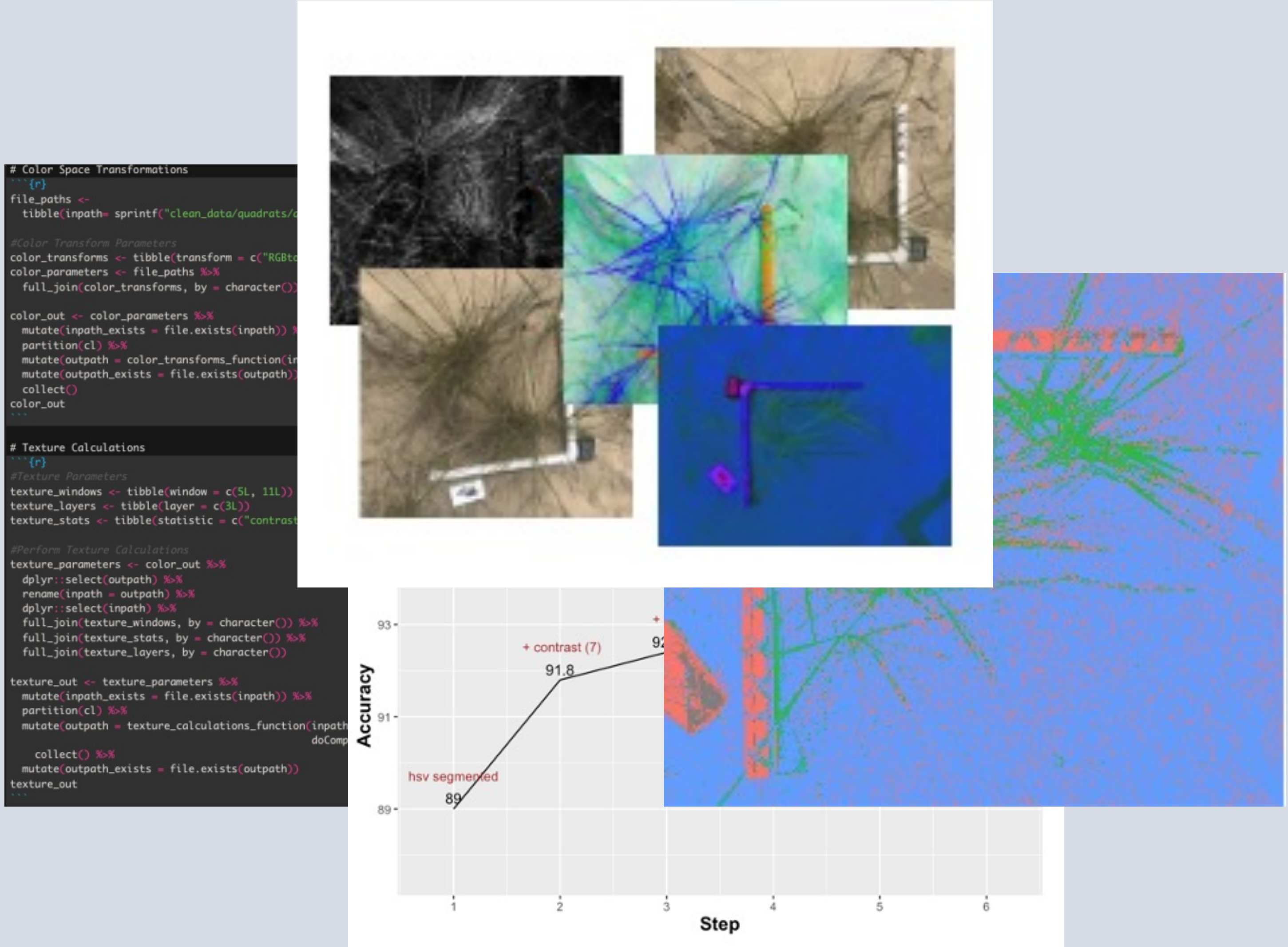
Technology

- RStudio, tidyverse
- Python,
- Labelme
- Linux

Code

View code [here](https://github.com/jackkrebsbach/classify-images)!

https://github.com/jackkrebsbach/classify-images



Modeling Dune Vegetation

Goal

- Generate vegetation density map across an entire coastal dune complex

Overview

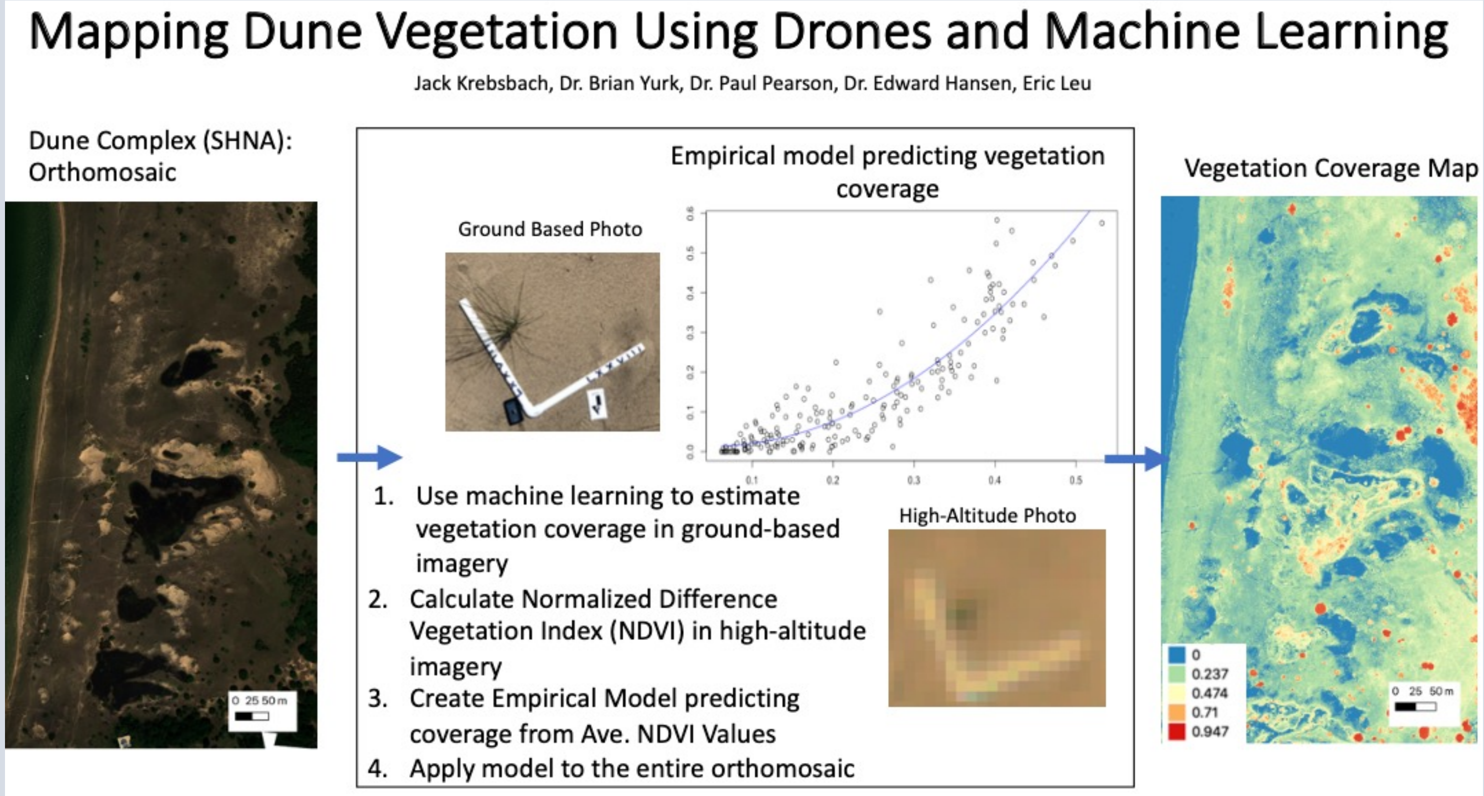
- Acquire initial vegetation estimates
- Create Normalized Difference Vegetation (NDVI) orthomosaic map
- Create model prediction live vegetation coverage from average NDVI values
- Calculate vegetation density o

Technology

- RStudio, tidyverse
- QGIS (Information Geographic System)
- Linux

Code

Code not yet available



Single Layer Neural Network

Goal

- Learn how Neural Networks works
- Implement a single layer neural network in python
- Classify ground cover in an orthomosaic acquired from a drone

Overview

- Acquire training set (QGIS)
- Implement a single layer ANN
  - Back propagation (gradient descent)
  - Activation Functions (Relu, SoftMax)
  - One hot encoding
- Classify orthomosaic
- Achieved 99% accuracy with 3+ classes

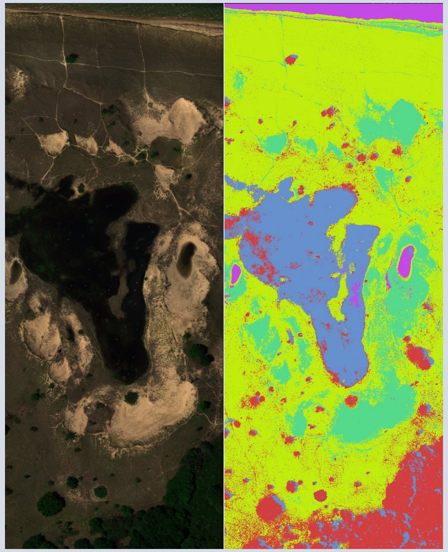
Technology

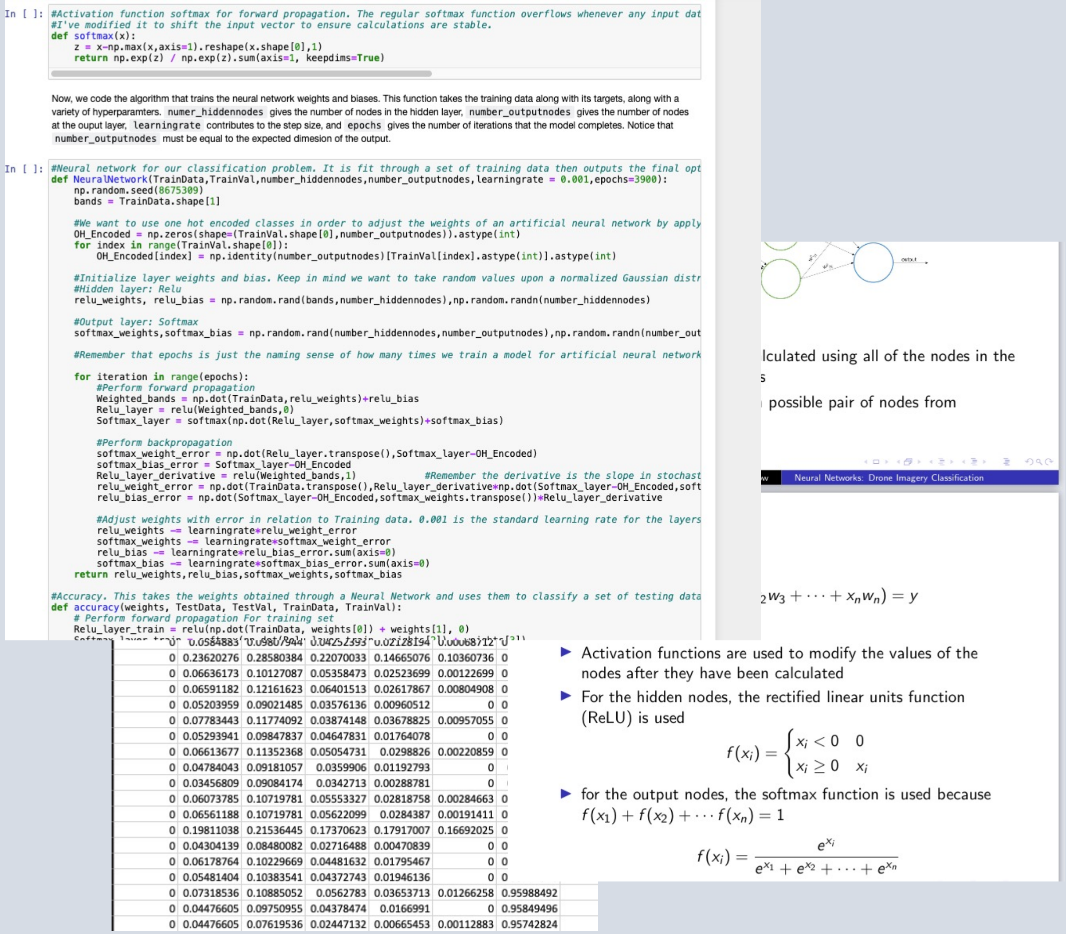
- Python, Anaconda
- Jupyter
- QGIS (Information Geographic System)

Code

View code here!

https://github.com/jackkrebsbach/neural-network





```
In [ ]: #Activation function softmax for forward propagation. The regular softmax function overflows whenever any input dat
        #I've modified it to shift the input vector to ensure calculations are stable.
        def softmax(x):
            z = x-np.max(x,axis=1).reshape(x.shape[0],1)
            return np.exp(z) / np.exp(z).sum(axis=1, keepdims=True)
```

Now, we code the algorithm that trains the neural network weights and biases. This function takes the training data along with its targets, along with a variety of hyperparamters. `numer_hiddennodes` gives the number of nodes in the hidden layer, `number_outputnodes` gives the number of nodes at the ouput layer, `learningrate` contributes to the step size, and `epochs` gives the number of iterations that the model completes. Notice that `number_outputnodes` must be equal to the expected dimesion of the output.

```
In [ ]: #Neural network for our classification problem. It is fit through a set of training data then outputs the final opt
        def NeuralNetwork(TrainData,TrainVal,number_hiddennodes,number_outputnodes,learningrate = 0.001,epochs=3900):
            np.random.seed(8675309)
            bands = TrainData.shape[1]

            #We want to use one hot encoded classes in order to adjust the weights of an artificial neural network by apply
            OH_Encoded = np.zeros(shape=(TrainVal.shape[0],number_outputnodes)).astype(int)
            for index in range(TrainVal.shape[0]):
                OH_Encoded[index] = np.identity(number_outputnodes)[TrainVal[index].astype(int)].astype(int)

            #Initialize layer weights and bias. Keep in mind we want to take random values upon a normalized Gaussian distr
            #Hidden layer: Relu
            relu_weights, relu_bias = np.random.rand(bands,number_hiddennodes),np.random.randn(number_hiddennodes)

            #Output layer: Softmax
            softmax_weights,softmax_bias = np.random.rand(number_hiddennodes,number_outputnodes),np.random.randn(number_out

            #Remember that epochs is just the naming sense of how many times we train a model for artificial neural network

            for iteration in range(epochs):
                #Perform forward propagation
                Weighted_bands = np.dot(TrainData,relu_weights)+relu_bias
                Relu_layer = relu(Weighted_bands,0)
                Softmax_layer = softmax(np.dot(Relu_layer,softmax_weights)+softmax_bias)

                #Perform backpropagation
                softmax_weight_error = np.dot(Relu_layer.transpose(),Softmax_layer-OH_Encoded)
                softmax_bias_error = Softmax_layer-OH_Encoded
                Relu_layer_derivative = relu(Weighted_bands,1)          #Remember the derivative is the slope in stochast
                relu_weight_error = np.dot(TrainData.transpose(),Relu_layer_derivative*np.dot(Softmax_layer-OH_Encoded,soft
                relu_bias_error = np.dot(Softmax_layer-OH_Encoded,softmax_weights.transpose())*Relu_layer_derivative

                #Adjust weights with error in relation to Training data. 0.001 is the standard learning rate for the layers
                relu_weights -= learningrate*relu_weight_error
                softmax_weights -= learningrate*softmax_weight_error
                relu_bias -= learningrate*relu_bias_error.sum(axis=0)
                softmax_bias -= learningrate*softmax_bias_error.sum(axis=0)
            return relu_weights,relu_bias,softmax_weights,softmax_bias

        #Accuracy. This takes the weights obtained through a Neural Network and uses them to classify a set of testing data
        def accuracy(weights, TestData, TestVal, TrainData, TrainVal):
            # Perform forward propagation For training set
            Relu_layer_train = relu(np.dot(TrainData, weights[0]) + weights[1], 0)
```

0	0.0584883	0.09807944	0.04252393	0.02128194	0.00068712	0
0	0.23620276	0.28580384	0.22070033	0.14665076	0.10360736	0
0	0.06636173	0.10127087	0.05358473	0.02523699	0.00122699	0
0	0.06591182	0.12161623	0.06401513	0.02617867	0.00804908	0
0	0.05203959	0.09021485	0.03576136	0.00960512	0	0
0	0.07783443	0.11774092	0.03874148	0.03678825	0.00957055	0
0	0.05293941	0.09847837	0.04647831	0.01764078	0	0
0	0.06613677	0.11352368	0.05054731	0.0298826	0.00220859	0
0	0.04784043	0.09181057	0.0359906	0.01192793	0	
0	0.03456809	0.09084174	0.0342713	0.00288781	0	
0	0.06073785	0.10719781	0.05553327	0.02818758	0.00284663	0
0	0.06561188	0.10719781	0.05622099	0.0284387	0.00191411	0
0	0.19811038	0.21536445	0.17370623	0.17917007	0.16692025	0
0	0.04304139	0.08480082	0.02716488	0.00470839	0	0
0	0.06178764	0.10229669	0.04481632	0.01795467	0	0
0	0.05481404	0.10383541	0.04372743	0.01946136	0	0
0	0.07318536	0.10885052	0.0562783	0.03653713	0.01266258	0.95988492
0	0.04476605	0.09750955	0.04378474	0.0166991	0	0.95849496
0	0.04476605	0.07619536	0.02447132	0.00665453	0.00112883	0.95742824

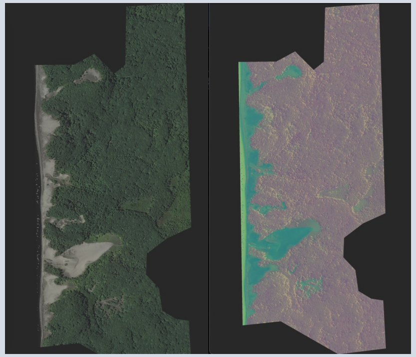
PCA Analysis

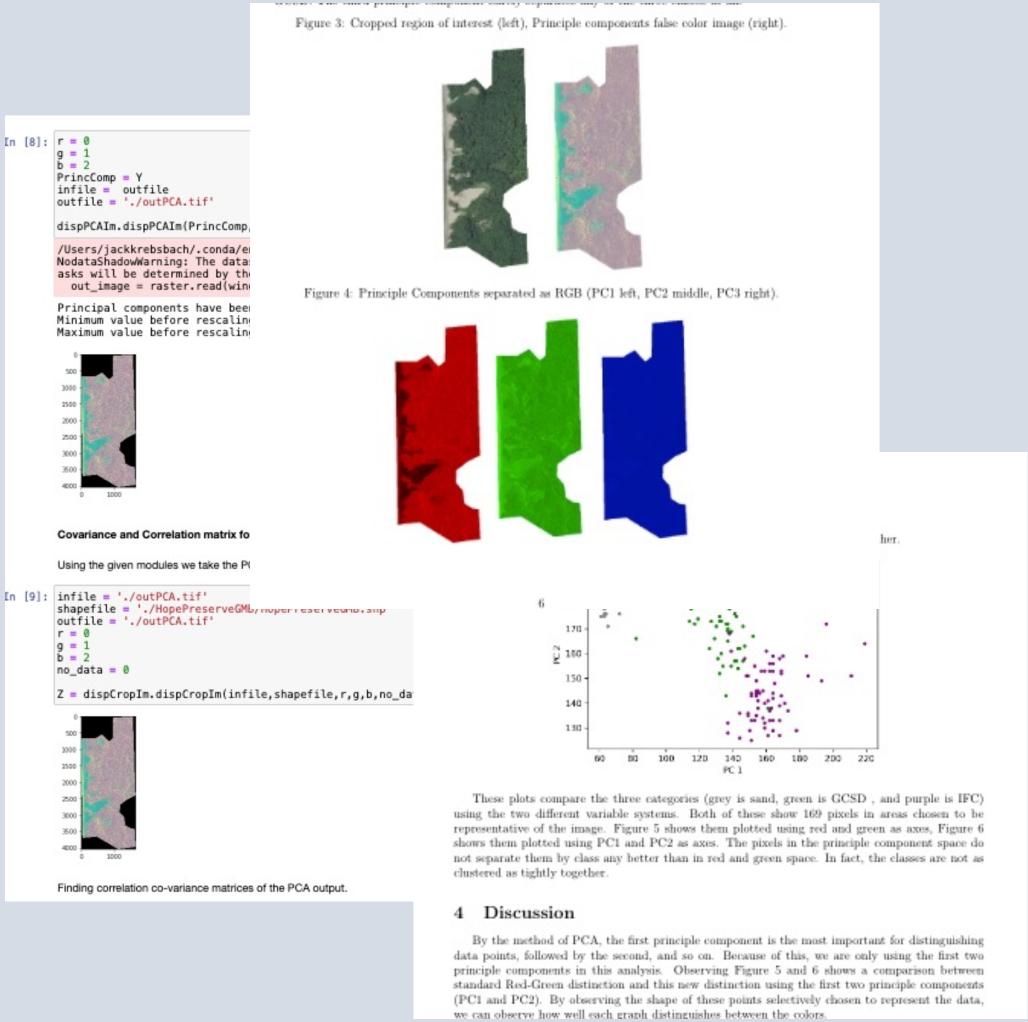
Goal

- Learn how Principal Components Analysis can be used
- Explore results of dimensionality reduction using PCA on RGB and NIR remote sensing data

Overview

- Acquire pixels of varying class from remote sensing data
- Plot original data
- Perform principal components analysis
    - Numpy for tensor manipulation
    - Eigh function to find Eigen Vectors
- Compare Principal Components with original data
- Write report using overleaf / LaTeX





Technology

- Python, Jupyter
- Rasterio, Numpy, Pandas
- QGIS (Information Geographic System)
- Overleaf / LaTeX

Code

View code here!

https://github.com/jackkrebsbach/principal-component-analysis