USING SPARSE CODING AS A PREPROCESSING TECHNIQUE FOR INSECT DETECTION IN PULSED LIDAR DATA

by

Connor Reece Zsidisin

A thesis submitted in partial fulfillment
of the requirements for the degree

of

Master of Science

in

Electrical Engineering

MONTANA STATE UNIVERSITY
Bozeman, Montana

December 2022

ACKNOWLEDGEMENTS

I would like to begin by thanking my advisor, Dr. Bradley Whitaker, for his guidance and instruction throughout my years at MSU. His help has been instrumental in my success in school. Dr. Whitaker has offered me many paramount opportunities in my career that will undoubtedly help me in my future as an engineer. I would also like to thank the remaining members of my committee: Dr. Rob Maher and Dr. Ross Snider for providing a first-class environment for learning. I would also like to thank Dr. David Dickensheets for being an outstanding advisor during my undergraduate degree, and Dr. Brock LaMeres for providing me with crucial opportunities as a teacher's assistant. Thank you to all these wonderful people for aiding me for the duration of my years here at Montana State University.

I would also like to thank Dr. Nathan Stark and the people who helped make the Spirit of the West marching band the best it could be. The marching band allowed me to make the closest friends in my life, and for that I am grateful. Band has allowed me to collaborate with a wide variety of people, and gave me the vital opportunity of becoming a section leader. My experience in the marching band helped me grow as a person and teach me lessons I will never forget. Ultimately, I would like to thank my family and friends for their support over the years. Their support through hard times, and especially COVID, has been immensely helpful.

This work was funded by the Air Force Research Laboratory (AFRL) on a subcontract from S2 Corporation under prime award No. FA8650-16-C-1954, with a fundamental research exemption.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

LIST OF FIGURES – CONTINUED

Figure Page

LIST OF FIGURES – CONTINUED

Figure Page

LIST OF FIGURES – CONTINUED

Figure Page

ABSTRACT

This research proposes using sparse coding as a preprocessing technique on insect lidar based data. This preprocessing technique will be used in conjunction with the Adaptive Boosting (AdaBoost), Random UnderSampling Boosting (RUSBoost), and neural network algorithms to automatically detect insects. The project aims to increase the effectiveness of these algorithms by using new images created by sparse coding. The K-Singular Value Decomposition (KSVD) algorithm will be used to train a dictionary on images that contain the majority class (non-insects). This trained dictionary will be used along with Orthogonal Matching Pursuit (OMP) to reconstruct all lidar images. The difference between the original image and the reconstructed image will be taken and processed by the feature extraction function and then used to train and test the models. Using a complete and an overcomplete dictionary our results show that the algorithms are able to detect insects at a higher rate. Using an overcomplete dictionary we are able to classify 93.18% of insect containing images in the testing dataset. Using the complete dictionary we were able to maintain 99.70% of non-insect images while increasing the percentage of insects classified to 84.09%.

1

INTRODUCTION

A healthy insect population is vital to the well-being of an ecosystem. Insects play many roles in an ecosystem, from being pollinators and detritivores, to being important food sources in the food chain [22]. With our ever-changing climate and, it is entomologists' goal to reliably predict the health and possible changes of an ecosystem by looking at data based on insect populations [21]. With the changing biodiversity of ecosystems [14, 35] it is becoming ever more important for entomologists to gather spatial distributions of insects in addition to sampling the populations [18]. The more efficiently entomologists can obtain this data, the faster they can identify issues in an environment and implement conservation techniques. Previous methods used to obtain data on insect populations include various traps and nets [3, 42, 28]. These techniques are invasive to the insect population and environment and time consuming when trying to spatially sample a population [18]. Certain techniques, such as pitfall traps, are likely to only catch nonflying insects and not entirely capture the local insect population. Lidar is a technique currently being utilized to observe flying insect populations non-invasively and spatially [20]. Wingbeat modulation pulsed lidar is one unique method for insect monitoring [31, 26, 30].

The goal of this work is to aid entomologists in effectively utilizing their time in the field and in the lab by automating the detection of insects in lidar images. This will eliminate the hours of time needed to sort through each individual lidar image and allow researchers to manage resources efficiently.

The data used in this project was collected from the Hyalite Creek area near Bozeman, Montana [38]. The pulsed lidar data was taken over a period of 4 evenings in September 2020. The visualization of this dataset and the previous train/test split is seen in Figure 1.1.

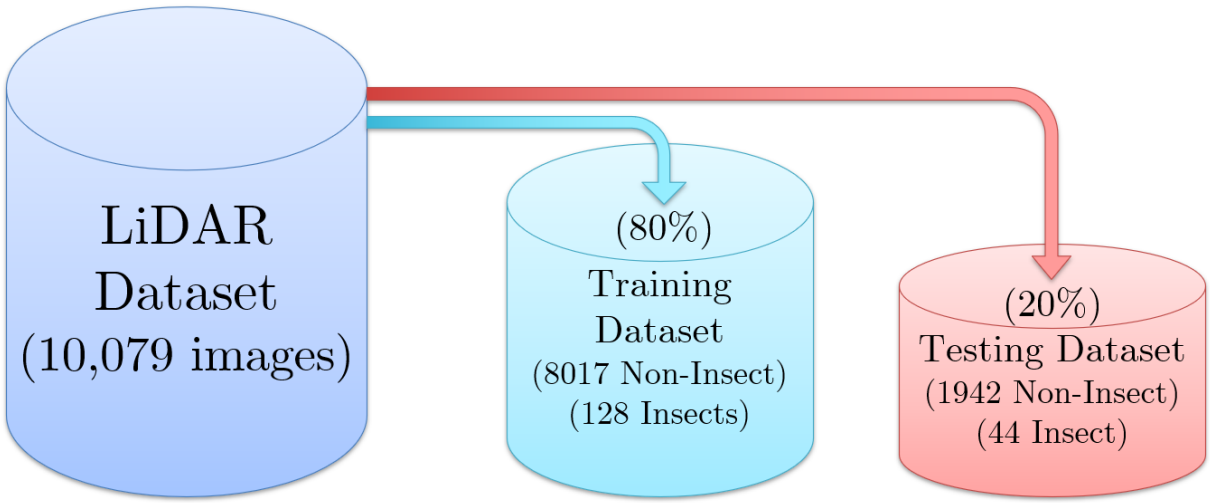

Figure 1.1: Dataset used in this work

This data was captured in a field environment to effectively train algorithms in insect classification. This research proposes the use of the sparse coding algorithm as a preprocessing technique to improve upon previous models. The sparse coding algorithm, K-Singular Value Decomposition (KSVD), will take a portion of the training data to train a dictionary. The atoms of this dictionary will be the learned features, or basis functions from the dataset. These features will be used in conjunction with a sparse vector, that has been limited to a certain number of non-zero entries, to generate a reconstructed version of the current image or signal. The sparse vector is optimized using the Orthogonal Matching Pursuit (OMP) algorithm [36, 4] to find the best atoms that can approximate the original signal. The difference between the original signal and the reconstructed signal is taken and fed into a feature extraction algorithm. The difference between the two images is taken to produce an image where the insect is more distinguishable. This image will follow the previously defined pipeline in the work of Vannoy et al. [40]. This pipeline will be feature in Figure 1.2.

KSVD is a tool used to implement sparse coding that is able to simultaneously train

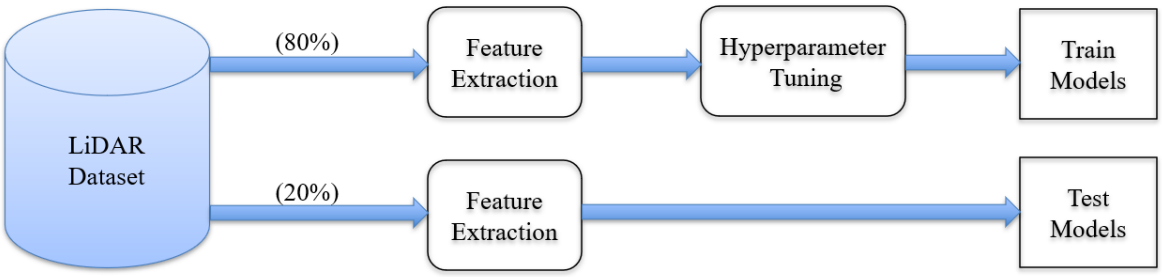

Figure 1.2: The previously defined pipeline from Vannoy et al.

a dictionary and optimize the sparse coefficients. KSVD allows for the optimization of the sparse representation of the input data, which will eliminate the insect and noise from the image. The difference image will have the feature extraction algorithm run on it to draw out important frequency and time features. The extracted features and the generated difference signal will then be used to train and test the previous models.

In summary, this thesis aims to improve upon past techniques by including a preprocessing stage to the previously defined pipeline. This preprocessing stage utilizes the sparse coding algorithm, KSVD, that will train three dictionaries capable of reconstructing the data set. This will result in improvements in insect detection and classification, and aid entomologists by automatically removing images that do not contain insects.

4

BACKGROUND

Insect LiDAR

Light Detection and Ranging (LiDAR) is a technique used for remote sensing [7]. Lidar is an approach that is comparable to radar, however instead of using radio waves to detect objects and estimate their range [9], laser light is used. Lidar is commonly used to capture three certain types of information. This information is spatial, spectral, and temporal [23]. Lidar instruments are commonly classified by these three types of information. However, every lidar instrument is capable of collecting spatial data as this is the foundational purpose of the device. Lidar works by taking time of flight measurements to obtain the spatial and temporal information. The lidar system that was used to collect data for this project is shown in Figure 2.1, this photo is provided courtesy of Trevor Vannoy. The photo seen in Figure 2.2 highlights the lidar system being used in the testing environment at Hyalite Creek, this photo is provided courtesy of Riley Logan.

Figure 2.1: Photographs of the lidar system used to collect the training and testing data. This photo is courtesy of Trevor Vannoy

Figure 2.2: Photographs of the lidar system at the data collection site. This photo is courtesy of Riley Logan

In lidar measurements spatial information can be collected using three styles of systems [23]. These lidar systems include the ability to find objects in one-dimensional systems, two-dimensional systems, and three-dimensional systems. The 2D and 3D systems are typically used in conjunction with optical deflecting systems [23]. These systems are able to create point clouds that are able to be used in a variety of applications from computer vision to surface reconstruction [41]. In these application spatial information is critical for reconstructing the environment. 3D mapping and topology is an active area of research in the remote sensing community and is being used for building detection and modeling [41]. In forestry, lidar is being used to measure canopy heights, reconstruct canopy topology, and above ground biomass. Lidar systems typically use the multiple returns from light pulses to estimate various structures of individual trees [2].

Spectral information in lidar systems is typically measured by measuring the laser return intensity (LRI) [23]. The reflectance of a material type is unique and can be used to identify a specific material type. Since LRI is unique and can be used to differentiate materials, researchers have been using it in a variety of ways. One such remote sensing application is being used by the USDA Forest Service [15]. The forest service is using the LRI to help distinguish and differentiate tree species in a forest. Their findings were promising and the results of the research aim to help manage and monitor our natural resources. Some of these applications involve wildfire risk assessment and wildlife habitat assessment.

Temporal information in lidar systems is gathered using the repeated lidar strategy. This technique is done by collecting data in a repeated fashion over a set amount of time. This can be used detect changes in an environment over that set amount of time. This data can be used to infer above ground biomass changes in forests [24]. Repeated lidar can also be used to track the changes in a forest due to erosion [25]. By using repeated lidar researchers were able to identify the dominant drivers of erosion after a wildfire has swept through an area. Using temporal information in lidar systems has allowed many researchers to view

changes of an environment, and highlights its effectiveness at modeling wide spans of area remotely.

Lidar is capable of capturing many meaningful aspects of an environment or object. By sending out laser light pulses systems are able to capture the spatial, spectral, and temporal pieces of information. Systems can measure how long light takes to return, the return intensity of that light, and the same environment multiple times. Using these key aspects of lidar, it can be applied to the detection and monitoring of insects and their populations. One method for monitoring insects with pulsed lidar is the utilization of wingbeat modulation. This technique utilizes the laser return intensity to obtain critical spectral information. As an insect flies through the pulsed lidar beam its wings introduce a modulated signal. This modulated signal is created by the changing the amount of light that is reflected back to the system. This modulated signal captures the insects and the frequency at which it beats its wings. This signal is time-limited as the insect does not tend to stay in the lidar pulse for an extended period of time. This data was taken over four evenings from the Hyalite Creek area near Bozeman, Montana and consists of 10,079 images. Each image contains 1024 pulses and 178 range bins, these are, respectively, the columns and rows in an image [38, 40]. Each row has a range resolution of 0.75 meters and each full column represents 200ms in time. This 200ms is comprised of 1024 individual light pulses that took approximately .195 milliseconds per pulse. An example of this image can be seen in Figure 2.3, this figure and caption were provided courtesy of Vannoy et al. [37].

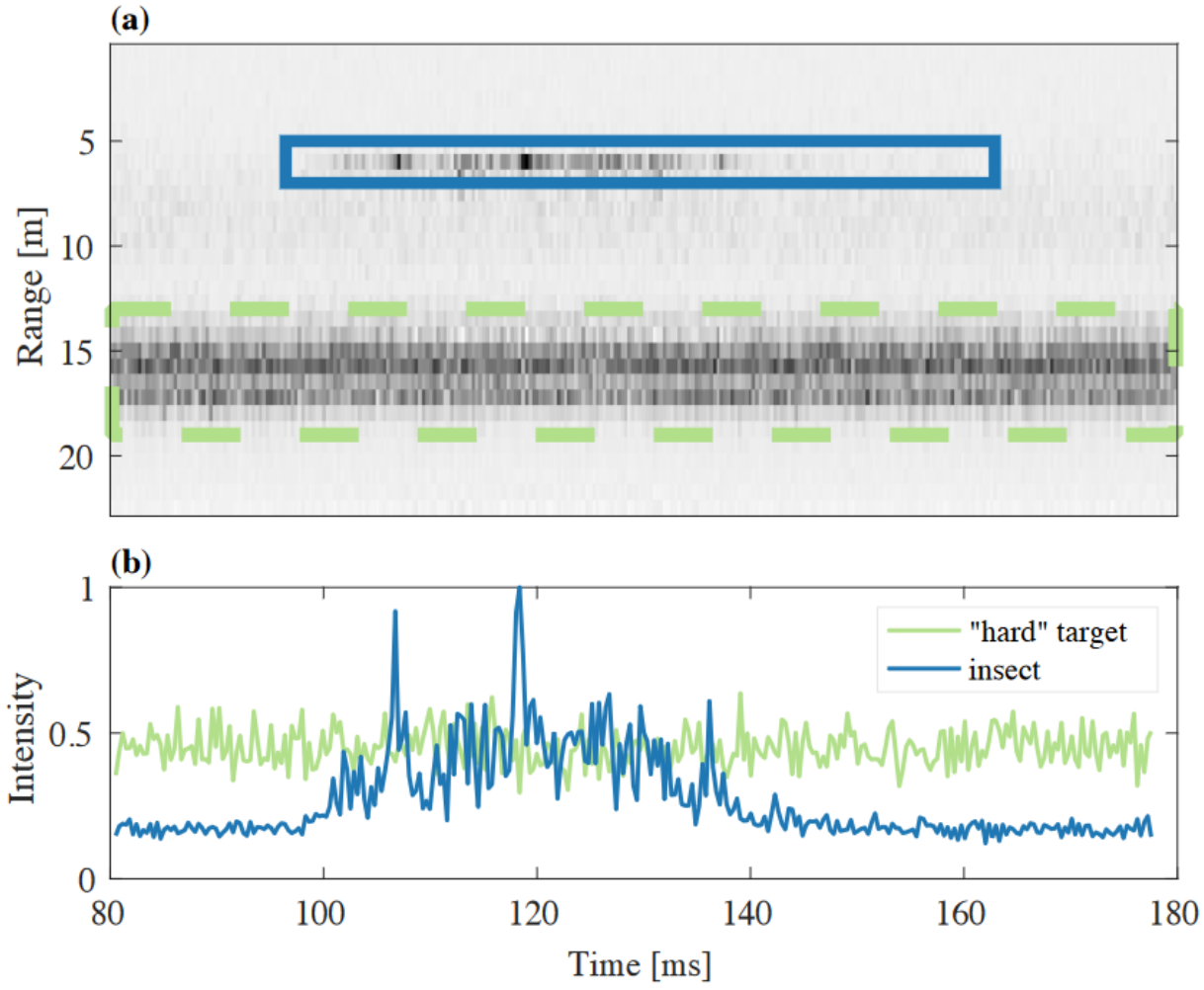

Figure 2.3: Example lidar image showing an insect and a "hard" target. (a) Range vs. time shown as a grayscale image. An insect, which is characterized by oscillations in intensity, is highlighted by the manually placed solid blue rectangle; a "hard" target, which could be a tree or shrub, is shown in the manually placed dashed green rectangle. (b) Comparison of the time–domain waveforms for the insect and the "hard" target; although the stationary object's response oscillates, the insect's waveform is easily distinguished by pronounced oscillations over a finite time frame, indicating that the insect flew through the lidar beam [37].

During the process of collecting data there are relatively few images that contain insects. Due to this, the dataset contains a large class imbalance. This poses an issue for the algorithms to correctly classify the desired class. To help solve this issue machine learning

algorithms need to be regularized, such that one class is not predicted in every instance.

Previous work aimed to solve this problem by including a feature extraction technique before training [40]. By extracting time and frequency domain features the algorithms can classify insects with higher precision. The previous models included a neural network [27], Adaptive Boosting algorithm (AdaBoost) [10, 11], Support Vector Machines (SVM) [6], and the Random UnderSampling and Boosting algorithm (RUSBoost) [29]. The majority of each individual lidar image contains either a hard target or noise. These portions of the data typically do not have interesting frequency domain features. The desired class (insects), do contain interesting time and frequency features as their wings introduce harmonic frequencies into the data. This interesting phenomenon allows the algorithms to learn the proper parameters capable of classifying the data.

The neural network algorithm [27] used in this project is a single layer neural network. This single layer is referred to as the hidden layer and can contain a variable number of nodes. Each of the individual nodes in the hidden layer are connected to each of the features in the input layer. These features are the from the observed data point and are used to find the optimal parameters for each node. Each node is connected to the output layer and gives the output layer a value. The output layer takes every value from the hidden layer to make a final classification on the observed data point. Nodes in the hidden layer use a nonlinear activation function to transform the data before the weights and biases are learned for each node. Each node takes the input features, transforms them non-linearly, makes a decision, and outputs a value. This outputted value is then used by the output layer using the softmax function to classify the current data point.

The AdaBoost algorithm [10] contains an ensemble of weak classifiers. Each weak linear classifier learns its own parameters in the feature space. After training each classifier, the congregate decision is made to make a string classifier. This algorithm depends on each weak learners performance. As long as the weak learners perform better than random guessing a

final strong classifier can emerge. The weak learners that are weighted the highest in the final decision are the learners that either get everything correct, or everything incorrect. The learners that are weighted the least are the weak classifiers that make a 50/50 decision. This method uses the congregate decision from an ensemble of weak learners for classification.

A Support Vector Machine [6] is a supervised linear classifier that aims to maximize the space between two classes with a dividing hyperplane. This hyperplane is placed in the margin, the place where the greatest separation between the two classes is calculated. This hyperplane can be tuned with a cost coefficient that helps determine how much each misclassification is weighted. This algorithm is a tunable linear classifier that aims to find the optimal separation between classes for classification. New examples are classified depending on where they fall in the feature space. There are two cases of the SVM that are typically used, the hard-margin SVM and the softy-margin SVM. The hard-margin variant is typically used with linearly separable data. This data can first be non-linearly transformed using a kernel, once its transformed the SVM linear classifier is then applied in this space. The soft-margin SVM is used for non-linearly separable data and involves the use of the hinge-loss function and the cost weight.

RUSBoost [29] follows the same general intuition as AdaBoost. It requires an ensemble of weak learners being used in conjunction to make a final decision. However, instead of using the entire dataset in each iteration to update the parameters of each learner, a smaller subset is used. In each training iteration random examples from the majority class in the dataset are removed to create a more balanced training environment for the weak learners. This may also be done in conjunction with oversampling, where extra examples of the minority class are added. Undersampling and oversampling are done until the desired class distribution is reached. Once this class distribution is reached the weak learners then make their decision and updated their weights and biases for every misclassified data point. This is done every iteration and will result in a strong classifier that may be better at identifying the classes in

a dataset.

A goal of this work was to merge a signal processing technique and apply it with machine learning pipeline previously used on insect lidar data. The chosen signal processing algorithm for the project was sparse coding. This algorithm learns and optimizes basis functions/features from the data and uses them to reconstruct the original data. In this process the original data is denoised, this is the preprocessing step for the work. This technique, combined with time and frequency feature extraction, processes the lidar images for the machine learning classifiers. A visual representation of sparse coding is represented in Figure 2.4 [5].

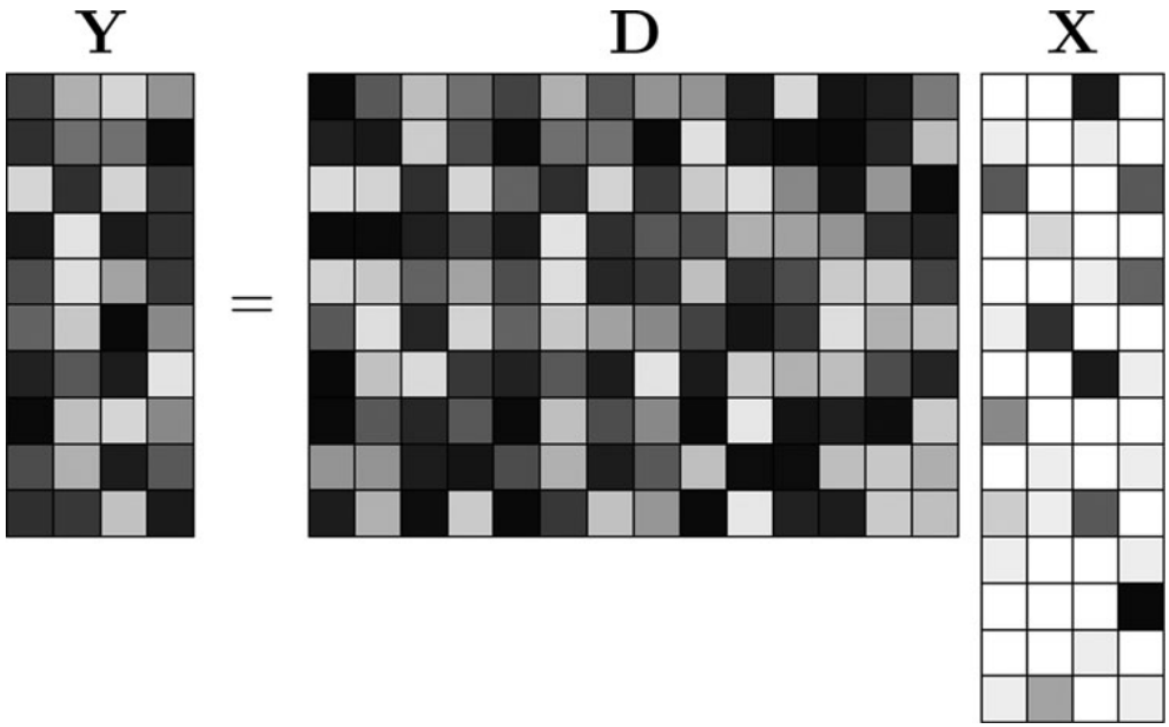

Figure 2.4: Visual representation of sparse coding. The feature matrix (left) is factored into the product of a dictionary matrix (center) and a sparse coefficient matrix (right) [5].

To the researcher's knowledge, sparse coding has not been used as a preprocessing technique for lidar data with the goal of classification. Sparse coding has been used with lidar for data restoration [19, 16] and denoising [13, 12]. Sparse coding is typically used for these applications in a wide variety of areas. Sparse coding is a common tool used to denoise data and fill gaps in data, in this work sparse coding is used for image denoising. These denoised images played a critical role in the results of the project.

Sparse Coding

Sparse Coding is a regularization technique that aims to represent signals as a linear combination of vectors. The goal of the technique is to find the hidden or underlying signal. The algorithm optimizes this representation under the constraint of only allowing a few nonzero coefficients. This new representation is known as the sparse representation of the input signal. These nonzero coefficients methodically select important features from a learned dictionary to reconstruct the original signal. There are three types of dictionaries that can be used with sparse coding. The sizes of these dictionaries depend on the input data, this is described in Figure 2.5. The input data resides in dimension N, in this work's dataset

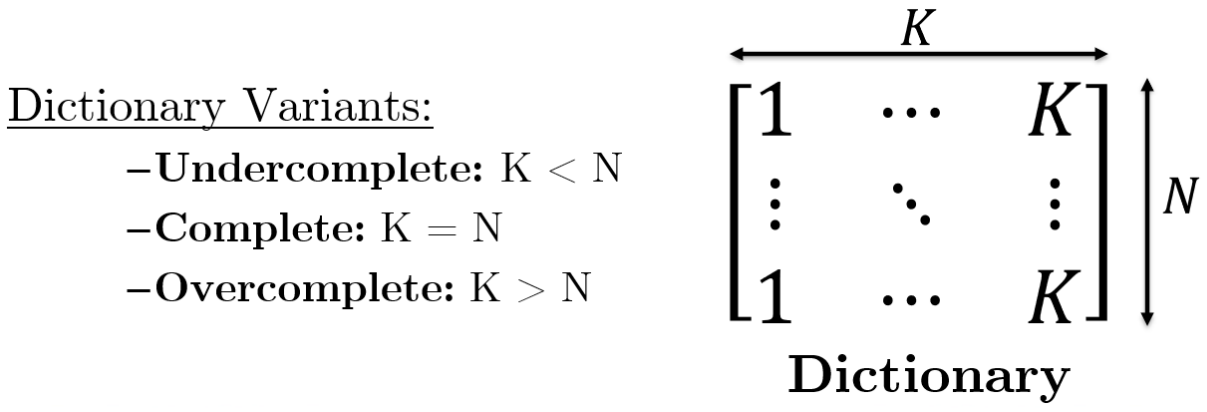

Figure 2.5: The variants of dictionaries used by this project

the input data used by the dictionary has a dimension on 1024. Each column of the row represents an individual light pulse sent out from the lidar system. Each row of an image represents approximately 200 ms in time as the lidar beam is swept through the environment. The first type of dictionary is an undercomplete dictionary. An undercomplete dictionary will have a size of NxK. In this work the undercomplete dictionary has a dimension K of 512, therefore the dictionary will be a matrix of size 1024x512. The complete dictionary variant will contain the same amount of columns as the dimension of the input data, this will be a matrix of size 1024x1024. The overcomplete dictionary will contain more columns than the dimension of the input data, in this research the dictionary is 1024x2048. These dictionary variants will be referenced as either the variant name or by the number of columns in the dictionary. For example, the undercomplete dictionary may also be referred to as D512.

Dictionaries are used alongside a sparse vector to reconstruct any input signal. This reconstructed version is known as a sparse representation. In this work the sparse representation of the observed row in any lidar image can be represented as 4 scaled atoms from a learned dictionary. This is visualized in Figure 2.6.

The process of learning the dictionary, D, and optimizing the sparse coefficients, $\bar{\alpha}$, can be done using the K-Singular Value Decomposition (KSVD) algorithm [1]. KSVD is an unsupervised, iterative approach to representing signals and learning features from data. The KSVD algorithm, in general, has two main steps, the sparse coding stage and the dictionary update stage. The sparse coding stage utilizes a matching pursuit algorithm [17] to find the best atoms in the current fixed dictionary to represent any given signal. The dictionary update stage keeps the established sparse coefficients and uses the Singular Value Decomposition (SVD) to update and optimize the current atoms in the dictionary. Matrix A holds every sparse vector, $\bar{\alpha}_i$, that is used to reconstruct the corresponding data, $\bar{x}_i$, from matrix X. This iterative process is repeated until the sparsity or error constraint is met; in this work the sparsity constraint is used. The algorithm is optimized using the following

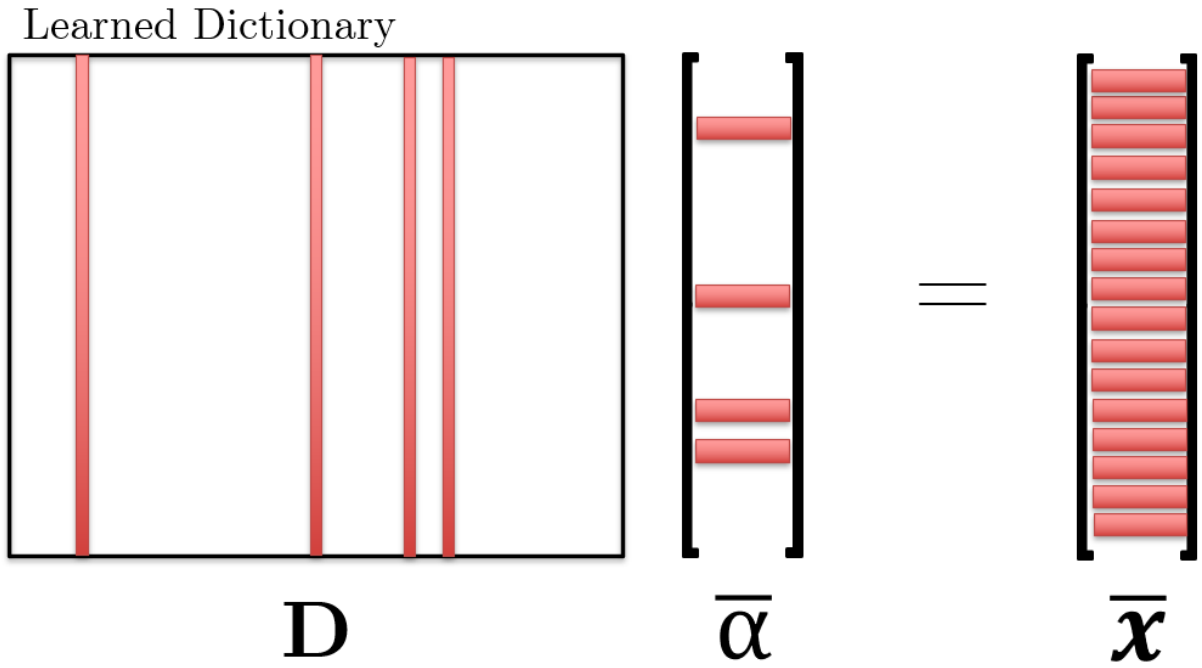

Figure 2.6: The sparse representation of an input signal. D is the learned Dictionary that the sparse coefficients select atoms from. $\bar{\alpha}$ is the sparse vector whose coefficients are optimized, and $\bar{x}$ is the resulting sparse representation.

equation (refer to Figure 2.6 for visualization of variables):

$$\min_{D,A} \|X - DA\|_F^2 \quad s.t \quad \forall i,\ \|\alpha_i\|_0 \leq k \qquad (Sparisty\ Based\ Constraint) \tag{2.1}$$

$$\min_{D,A} \sum_i \|x_i\|_0 \quad s.t \quad \forall i,\ \|X - DA\|_F^2 \leq \epsilon \qquad (Error\ Based\ Constraint) \tag{2.2}$$

KSVD: Sparse Coding Stage

The sparse coding stage utilizes a pursuit algorithm. The algorithm typically used in this application is a matching pursuit algorithm [17] that finds the optimal atoms from either

a learned or redundant dictionary.

$$x \approx \hat{x} := \sum_{n=1}^{N} \alpha_n d_n$$

The original signal, x, is approximated using a finite number of atoms from the fixed dictionary. The project uses a constraint of four non-zero coefficients, therefore N will be equal to four. The α_n is the scaling factor of each atom that optimizes our approximation of the original signal, known as our sparse coefficients. These scalar values are updated later using the least squares approach. The atoms that the matching pursuit algorithm chooses to approximate the original signal are represented by $\bar{d}_n$, these atoms reside in the dictionary, D. The combination of $\alpha_n \bar{d}_n$ will create a scaled version of an atom, which is a learned feature from the training data. Once the four representations of the original signal are computed, they are linearly combined to create our approximation of the input signal, $\hat{x}$. This will reconstruction will be known as the "sparse representation" of the signal. Each row will be constructed using only four atoms.

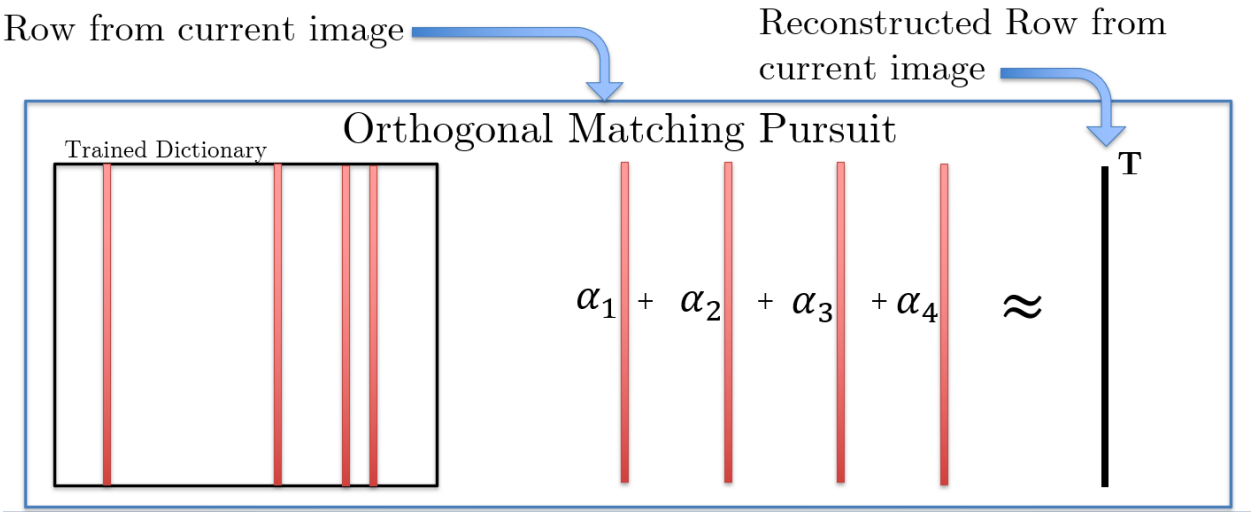

Figure 2.7: Visual representation of OMP

Using the overcomplete dictionary this means that only 4 atoms out of 2048 atoms will be used to reconstruct any given row. $\bar{\alpha_n}$ will be a sparse vector, in that it has 4 nonzero values and 2044 coefficients that correspond to zero. Each sparse representation will be stored correspondingly until we have our final approximated lidar image $\hat{X}$, previously shown as DA. Where, A, holds the sparse vector for each observed signal, and D is the dictionary that the atoms are selected from. The end goal of this technique is to minimize the residual error under the constraint of a few nonzero coefficients:

$$\min\|X - \hat{X}\|^2 \quad s.t. \quad \forall i,\ \|\alpha_i\|_0 \leq k \tag{2.3}$$

The algorithm used in this project was the Orthogonal Matching Pursuit (OMP) algorithm [43]. The algorithm first evaluates all atoms in the fixed dictionary and calculates the atom that best matches the original signal. If this atom meets the sparsity or error-based threshold the algorithm ends. If the threshold is not met, then the next best atom that minimizes the residual is chosen. Coefficients for these atoms are re-evaluated every iteration by using least squares to get a more efficient solution.

$$\tilde{\alpha} := \arg\min_{z \in \mathbb{R}^i} \|x - D^{(i)} z\|_2^2 \tag{2.4}$$

In this case $D^{(i)}$ represents the relevant columns of the dictionary and z is the least squares solution to the relevant signal, x. This process repeats until the residual is sufficiently small. Each signal has its own sparse representation and can be compared to the original. Once the best sparse coefficients and atoms have been chosen for each signal, the results are then given to the dictionary update stage.

KSVD: Dictionary Update Stage

The sparse coefficients, in the matrix A, found in the previous stage are fixed in place and the dictionary is systematically updated by checking each atom from the dictionary. The atoms of the dictionary to be updated are updated one at a time. It is updated by using the signals that have chosen the particular atom and used it for reconstruction. If the k-th atom is used by five signals, then those five examples are used to update the k-th atom. Before updating the atoms of the dictionary, the residual matrix must first be formed from the examples using the k-th atom. To find the residual error, an outer product is formed between the relevant sparse coefficients and the k-th atom. This is turned into a minimization problem to minimize the residual error and update the k-th column in D.

$$\min\|E_k - \bar{\alpha} D_k^T\| \tag{2.5}$$

The update is then performed by using Singular Value Decomposition (SVD) [33] on the resulting matrix. This update is done by multiplying $E_k X_k^T$, and then computing the singular value decomposition on this matrix. The updated atom is found by: $D_k = UV^T$ [1]. The optimized and updated atoms are then stored, and the next atom is chosen. The process repeats for each atom in the dictionary. The new dictionary is then fixed, and the sparse coding stage begins anew. This iterative process is repeated until the sparsity or error constraint is met.

METHODS

This work will use the pipeline previously utilized by Vannoy et al. [40]. This previous pipeline involves four major components. The data first goes through a feature extraction process that finds important time and frequency features. The extracted time features include the following [40]:

- The image mean subtracted from the observation mean
- The standard deviation of each observation
- The maximum absolute first difference

The frequency domain had many features extracted in this process. The spectral information that the lidar system provides is paramount in the detection of insects in lidar data. Oscillations induced by the insect in lidar data aids in separating it from non-oscillating hard targets. These oscillations provide important frequencies and harmonics that are extracted from the data. The features extracted from the frequency domain came from the observation's one-sided power spectral density [40] and consist of the observation's:

- Mean
- Standard deviation
- Median
- Median absolute deviation
- Mean-centered skewedness
- Mean-centered kurtosis

Once the fundamental frequencies were computed, the features extracted for each harmonic consist of each harmonic's:

- Height
- Location
- Peak Prominence
- Half-prominence peak width
- Ratios of the heights for all combinations of harmonics
- Ratios of the widths for all combinations of harmonics
- Ratios of the prominences for all combinations of harmonics

Following the feature extraction process is the hyperparameter tuning. Once the hyperparameters for each model are tuned, the models are then trained. These trained models are then used with the testing dataset to get the testing results. These results are valuable as the models are able to be used with data that has not been previously seen and their performance can be measured. This project introduces a preprocessing stage that takes place before the feature extraction process. This will be the sparse coding stage of the pipeline and this will be where the dictionary is trained and the optimal sparse representations are constructed. These reconstructed representations will be used alongside the original dataset to generate difference images. These difference images will replace the original images and will follow the same pipeline. The preprocessing stage is visualized in Figure 3.1.

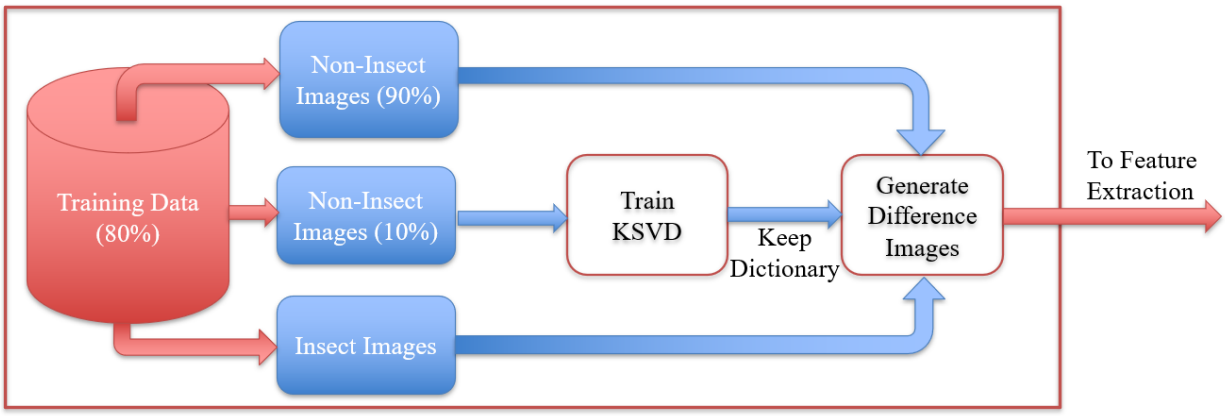

Figure 3.1: Visual representation of the sparse coding preprocessing stage.

To use sparse coding on the lidar dataset, a dictionary must first be trained on a subset of images. To increase the robustness of the dictionary, and to help highlight insects in the resulting images, it must be trained on 10% of the non-insect portion of the training data. This dictionary is trained on only non-insect data such that the resulting atoms can only reliably reconstruct features inherent to the background data. Sparse coding is very valuable in representing low dimensional data, this was previously seen with image denoising [8]. Insects are high dimensional in nature and therefore are hard to represent using a select number of features. Due to the high dimensionality of insects and the dictionary being trained on non-insect data, this creates an environment that is not conducive to reconstructing data containing insects. This environment is used as an advantage in preprocessing data. This is all described in Figure 3.2.

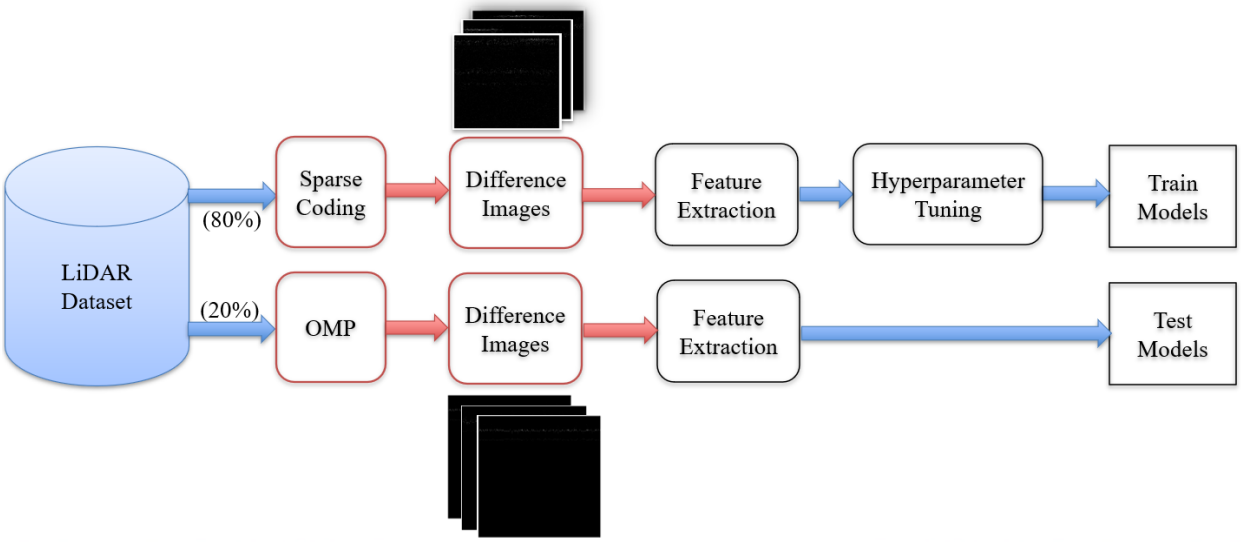

Figure 3.2: Revised Pipeline for insect detection

Dictionary Training

The training dataset contains 8017 non-insect images and 128 insect images. 801 non-insect images were chosen randomly from the dataset to be used as training examples for KSVD. These images were used to train 3 different dictionaries to test the effectiveness of each variant. Each dictionary's training examples were individual rows from each image. Each row from the lidar images will be treated as a data point. The observed row contains 1024 columns of information as the lidar beam is pulsed through the environment. The sparsity-based optimization for KSVD was chosen allowing for only 4 sparse coefficients. The dictionary was initialized using the Overcomplete Discrete Cosine Transform (ODCT) [1, 34]. The dictionary dimensions of each variant were 512, 1024, and 2048. An undercomplete, a complete, and an overcomplete dictionary were each used to preprocess the data before training on models. Using a smaller number of sparse coefficients is done to denoise the images and prevent the dictionary from reconstructing insects properly.

KSVD was run for 200 iterations as seen in Figure 3.3. As the dictionary atoms and sparse coefficients were optimized during the training of KSVD, the root mean square error converges to approximately .2825. This convergence occurs after roughly 60-80 iterations in the overcomplete dictionary case. Running KSVD for longer than 200 iterations caused minimal gains, even after 100 iterations minimal gains were observed. Since minimal gains were observed in the last 100 iterations the decision was made to not extend the 200 iteration count.

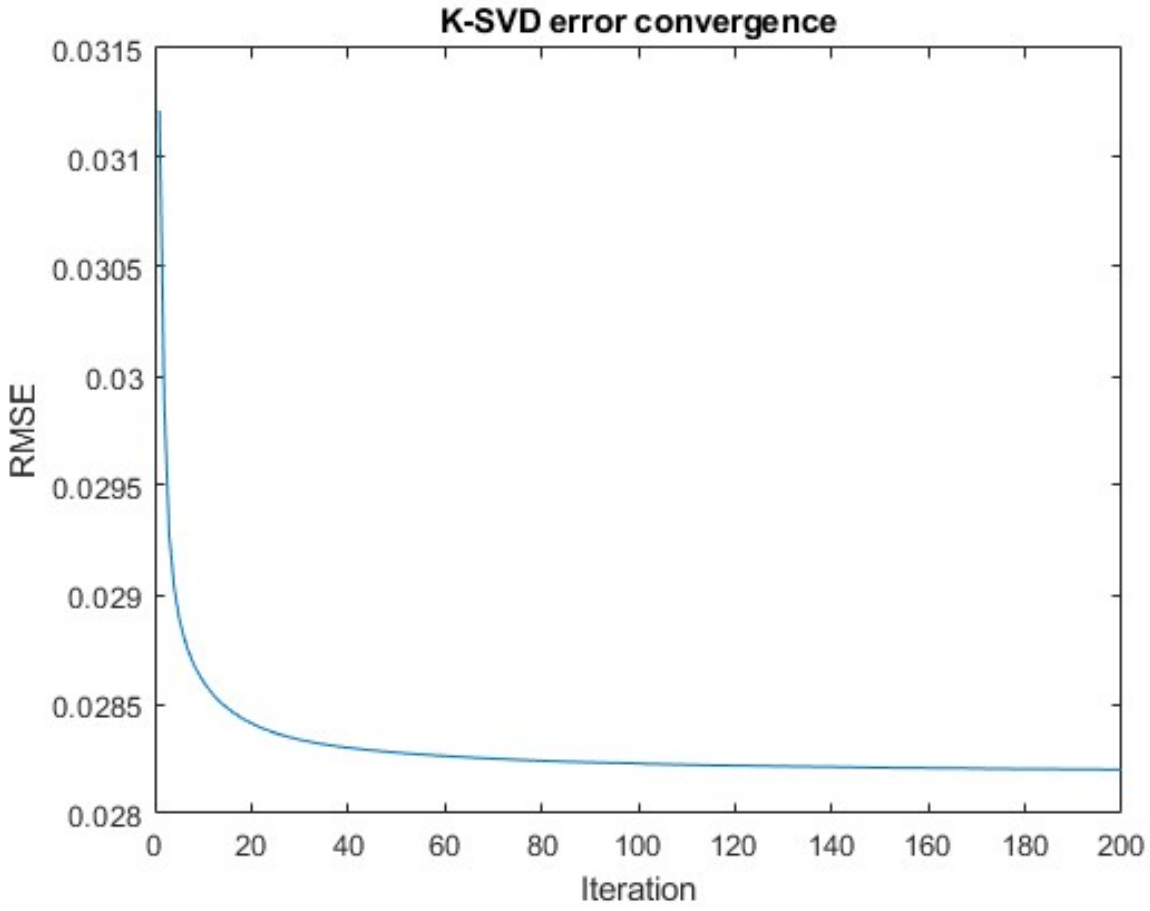

Figure 3.3: KSVD RMSE Convergence

Generating Difference Images

After finalizing the learned dictionary, OMP was run using the dictionary on the rest of the training data and the testing data. This step generates the reconstructed images of the dataset. The difference between the original images and the reconstructed images is taken. In Figure 3.4 the original (a,d), reconstructed (b,e), and difference (c,f) images are displayed. The bottom row of images (d,e,f) demonstrate how non-insect images are processed throughout the life cycle of the program. The top row (a,b,c) highlights how an image containing an insect is treated. The white box in the image, placed manually, highlights the insect that was captured in the image. The learned dictionary, combined with the constraint of 4 sparse coefficients cannot properly reconstruct the insect and noise in the data. The reconstructed images show the underlying hard targets and their reflections in their range boxes without noise. This is the latent distribution of the lidar data. The difference images show the added observed noise and the insect being removed from the reconstruction.

The dictionary and OMP treat the insect as high dimensional noise and remove it when reconstructing the image. This result is useful in that the difference image can be used as data for the previous models to train on. Most of the image will remain unchanged as it contains empty space or a hard target that the dictionary has been trained to reconstruct. The position of the insect in the image had no affect on the algorithms ability to reconstruct it. A few example images can be seen in Figure C.8 and C.10, the insects position varies in each of these difference images and the hard target is removed each time. The hard targets position also had no affect on OMP's ability to reconstruct it. This can also be seen in the difference images, as the noise that surrounds the hard target can still be observed. The main differences between the original image and the reconstructed images are insects and noise generated in the lidar measurement. This allows the feature extraction algorithm to

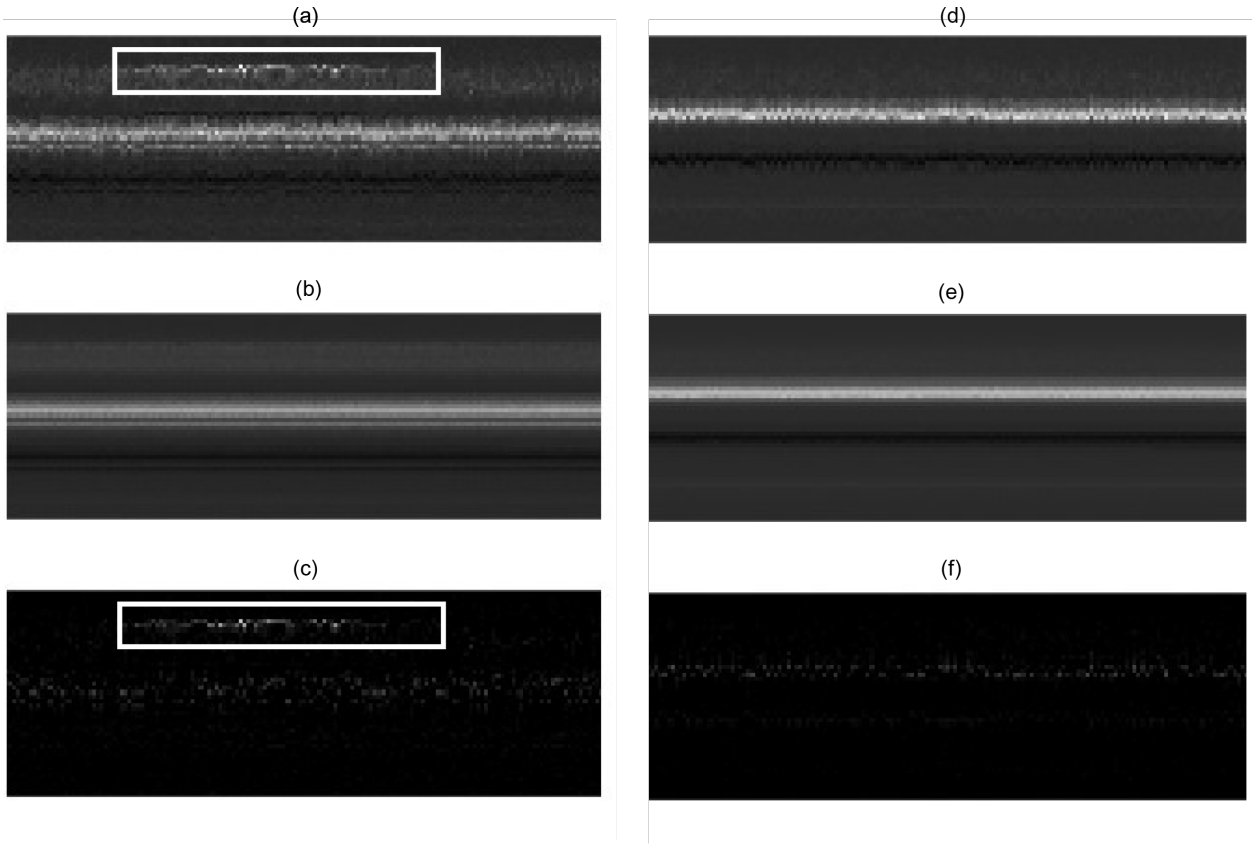

Figure 3.4: Examples of input images, reconstructed images, and difference images: (a) Original Insect Image (b) Reconstructed Insect Image (c) Insect Difference Image (d) Original Non-Insect Image (e) Reconstructed Non-Insect Image (f) Non-Insect Difference Image. All images are 256-bit grayscale (black = 0 and white = 255). Note that the reconstruction of the insect image does not include the image, so the insect is preserved in the difference image. The white box highlighting the insect was placed manually.

more efficiently find meaningful time and frequency features of the insects in the data. More examples of difference images can be viewed in Appendix B, 5 non-insect images and 5 insect images can be observed.

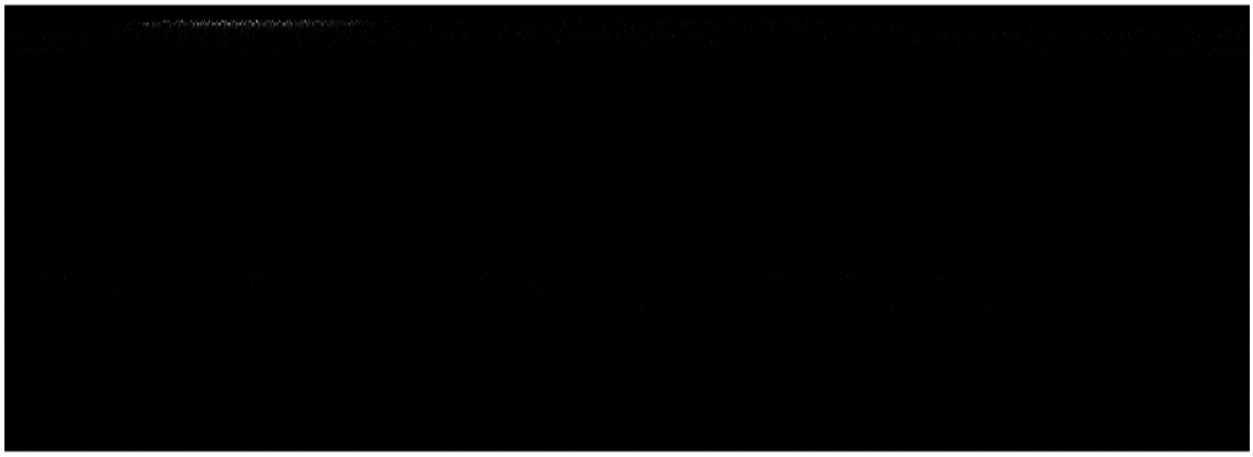

Figure 3.5: Example of a difference image that contains an insect.

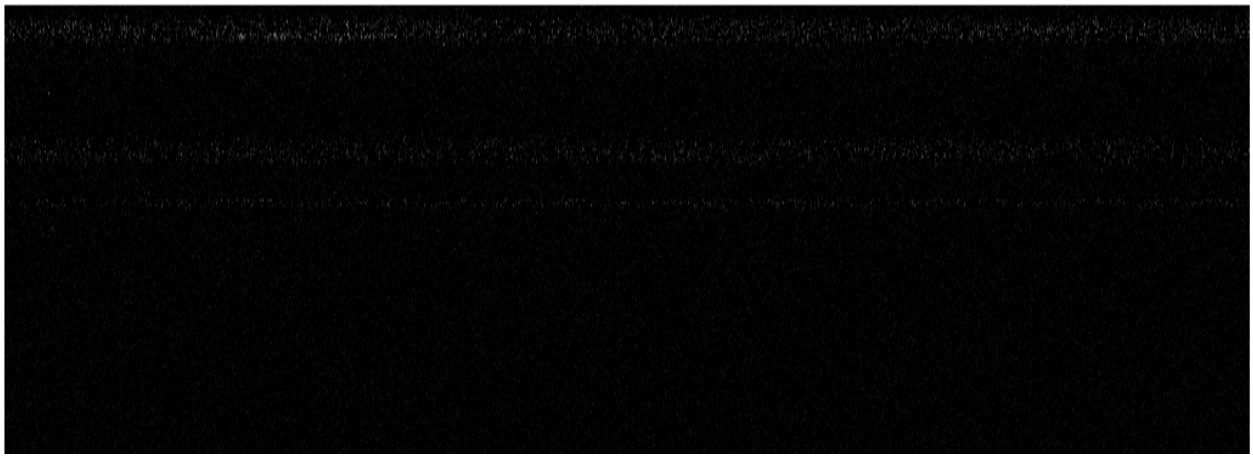

Figure 3.6: Another example of a difference image that contains an insect.

This sparse coding and difference image portion of the pipeline concludes the preprocessing for the training portion of the work. Using the trained dictionaries this is performed again with the testing set of data. OMP and the trained dictionaries are used on the testing set to generate reconstructed versions. The difference images are then generated using the same process of subtracting the reconstructed image from its doppelganger in the untouched testing data. This difference image is again run through the feature extraction process. These

extracted features and difference images now await the completed training of the classifiers.

Model Training

To properly train the AdaBoost, RUSBoost, and Neural Network algorithms, the data needs to run through a feature extraction process. This process extracts important time and frequency features that the models use to help identify insects. The produced difference images are fed into this process and the resulting features are extracted. Applying difference images with the feature extraction increases the insect detection rate by 28.11%. Noise has no meaningful features as it is gaussian and does not provide any useful information in the frequency domain. Insects have important frequency domain features that are captured by the lidar measurement due to the insect and its wings beating in the measurement. This allows the features of the insects to be further separated from the non-insect features.

The extracted features form the feature space that the machine learning classifiers work in. In this feature space each classifier will be trained and will learn the optimal parameters to classify the data. To ensure that classifier is given the best tools possible a few tuning algorithms are first run [40].

The first of the tuning algorithms is used to find the undersampling ratio and the number of synthetic insects; these are required to train the final models. This algorithm performed an extensive grid search to find the optimal undersampling ratio and the optimal number of synthetic insects. The role of the synthetic insects was to increase the number of insect examples for each classifier. This data augmentation is important in the training process as the more examples that the classifiers see of insects, the more effective they are at distinguishing an image containing an insect. The undersampling ratio is used to randomly sample the majority class so that the resulting dataset for the classifier has the desired ratio of majority and minority classes.

The second of the tuning algorithms used pertains to the hyperparameters of each

model. These hyper parameters can be tweaked to change the performance of the algorithm. For example, the neural network has hyperparameters such as the number of nodes in a layer and the activation function used with that layer. The sampling ratios and hyperparameters were assumed to be relatively independent [40]. The hyperparameters were tuned by using Bayesian optimization [32]. In this stage, and the previous, the MCC score was used to evaluate the performance of the algorithm. Once the tuning algorithm finishes, the best hyperparameters are saved and used to train the relevant models. An example of the results from the hyperparameter tuning stage can be seen in Figure 3.7. The hyperparameters are used to evaluate an objective function and show the improvements being found through the search. This was done using MATLAB's bayesopt function, with the setting of "expected-improvement-plus" function.

Finally, after the setup is complete for each classifier, training is able to begin. The training step of the pipeline is performed by using a stratified 5-fold cross validation of the training set [40]. This allows the individual classifier to see the same data multiple times and relay a cross validation score for the hold out set during training. This score highlights the classifiers predicted performance on testing set, even if it is optimistically biased. Once training is completed the trained classifier is then saved and stored to be used later during the testing phase.

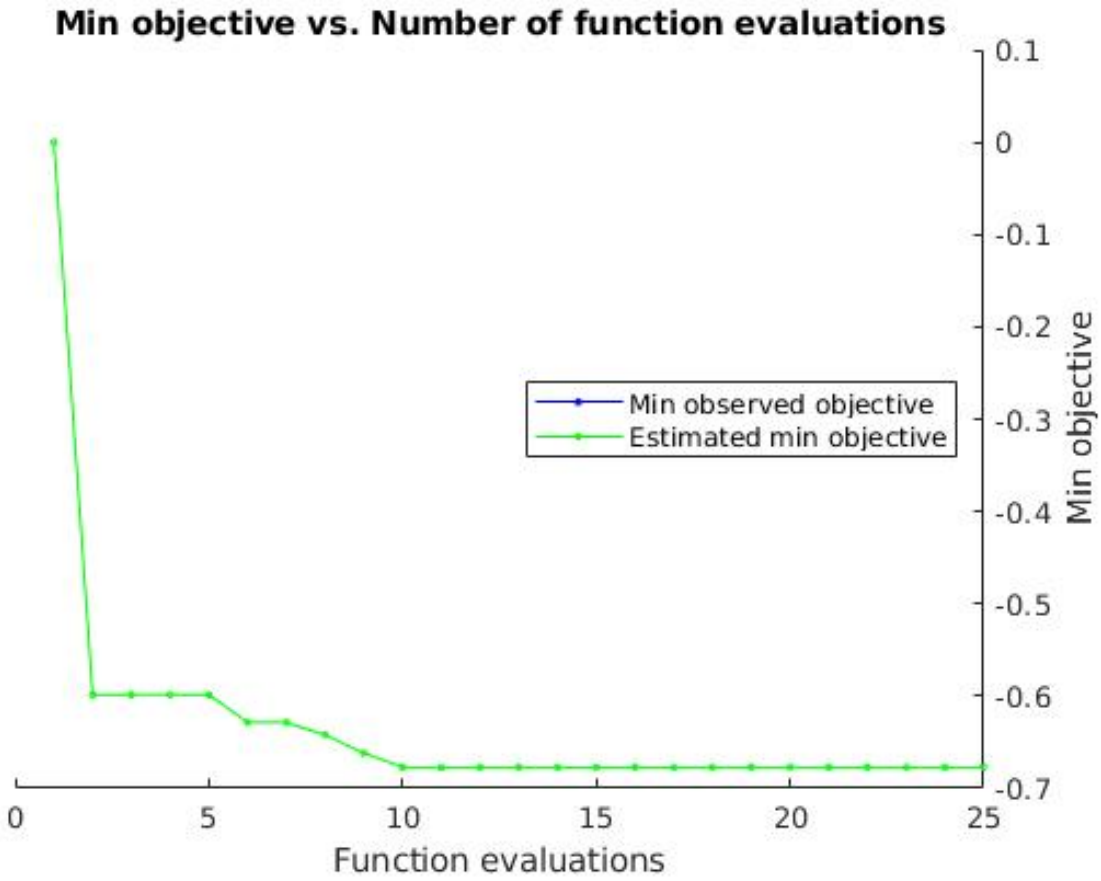

Figure 3.7: Hyperparameter Tuning Evaluations for RUSBoost using the Overcomplete Dictionary

Model testing

Properly testing the models is done by using the trained classifiers and the previously preprocessed data. The previously extracted features and difference images are used by the classifiers to generate the results. These classifiers will attempt to classify the data to the best of their ability and will generate a confusion matrix once finished. This confusion matrix keeps track of the correctly classified and misclassified observations and images. The result that we are interested in is the classifier's performance on images. This performance

is scored by extracting the 5 ratings discussed later. These scores include the classifiers accuracy, precision, recall rate, the F2 score, and the Matthew's Correlation Coefficient.

This work introduces a preprocessing stage into the previously defined pipeline. This preprocessing stage makes the use of sparse coding to train a dictionary that learns important features from data. This dictionary is trained by using the rows of an image as an input and the dictionary is optimized to reconstruct any observed row. These observed rows contain hard targets, the lidar artifacts of the hard target, noise, and empty space. The images containing insects are excluded from the dictionary training process to increase the robustness of the dictionary. This trained dictionary is then used with OMP to reconstruct every remaining image in the training dataset. This reconstructed image is then subtracted from its counterpart in the original dataset. This difference image then follows the pipeline. Difference images first go through a feature extraction process and are then used to tune the sampling ratios and the hyperparameters of each classifier. The classifiers: AdaBoost, RUSBoost, and a single layer neural network, are then trained using the extracted features and their optimal hyperparameters. Each classifier was then trained using a stratified 5-fold cross validation of the training set [40]. Once training has completed, the model was stored as a compact classifier and then used on the preprocessed testing set of data. The following results, and their discussion, come from the classifier's performance on the preprocessed testing set.

RESULTS

The results from this project are compared to the previous results of Vannoy et al.; to prepare these results for comparison we used the newest version of the code available on GitHub [39, 40]. In this work, the preprocessing was performed using a dictionary with different dimensions. The results for the undercomplete, complete, and overcomplete dictionaries will be compared to the best results of Vannoy et al. The important metrics that are to be analyzed are the 5 scores given to each algorithm for its performance with varying dictionaries. Figure 4.1 highlights the different confusion matrices. The previous AdaBoost confusion matrix (a), can be compared to the two differing RUSBoost matrices. The results of RUSBoost using the complete dictionary are shown in (b) and using the overcomplete dictionary are shown in (c). The final tuned hyperparameter values for each classifier can be seen in Tables 4.3, 4.4, and 4.5.

Table 4.1: Test Set Image Results

	D512	D1024	D2048	[40]
AdaBoost				
Accuracy	0.9933	**0.9943**	0.9938	0.9917
Precision	0.8974	**0.9714**	0.8810	0.8889
Recall	0.7955	0.7727	0.8409	0.7273
F2	0.8140	0.8057	0.8486	0.7547
MCC	0.8415	0.8638	0.8575	0.8000
RUSBoost				
Accuracy	0.9902	**0.9943**	0.9912	0.9840
Precision	0.8788	0.9024	0.7455	0.6066
Recall	0.6591	0.8409	**0.9318**	0.8409
F2	0.6938	0.8525	**0.8874**	0.7806
MCC	0.7564	**0.8682**	0.8292	0.7066
NNET				
Accuracy	0.9891	0.9871	0.9866	0.9907
Precision	0.8108	0.8519	0.8000	0.9333
Recall	0.6818	0.5227	0.5455	0.6364
F2	0.7042	0.5665	0.5825	0.6796
MCC	0.7381	0.6616	0.6543	0.7666

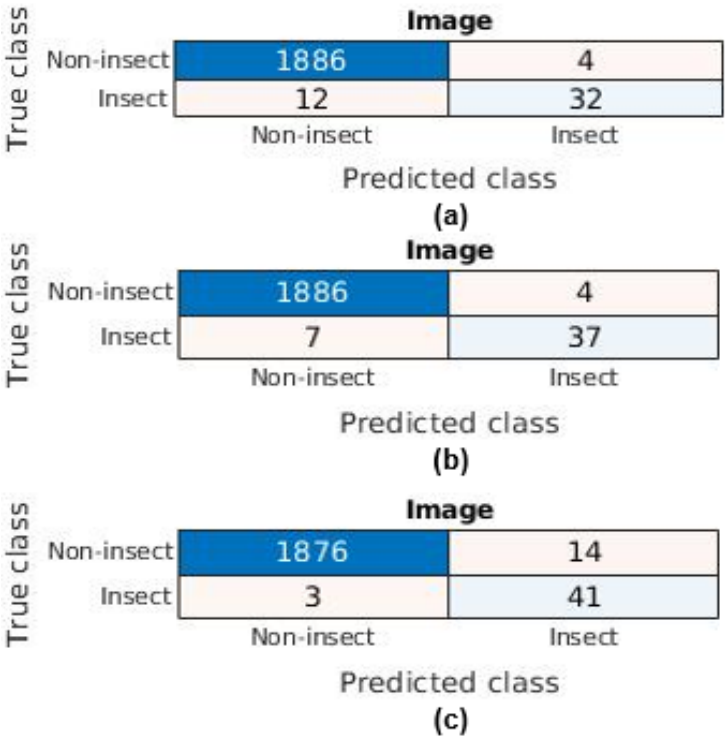

Figure 4.1: (a) Previous AdaBoost Confusion Matrix [39] (b) RUSBoost using Complete (1024) Dictionary (c) RUSBoost using Overcomplete (2048) Dictionary

The test results can also be compared to the cross validation results found during the training portion of the project. Cross validation results were found using a 5-fold stratified approach that was discussed earlier. The cross validation results are found in Table 4.2.

High scores in individual metrics are highlighted in bold for the cross validation results. The highest scores obtained in the cross validation stage mostly mirror the final test results that were found. The only high score difference between the two results lies in the RUSBoost algorithm preprocessed with the complete dictionary. In the cross validation stage it did not score the highest accuracy and fell short by .0001, or .01%. However, this version still maintained the highest MCC score during the cross validation stage. The RUSBoost algorithm that was preprocessed by the overcomplete dictionary had the highest recall and

Table 4.2: Cross Validation Image Results

	D512	D1024	D2048
AdaBoost			
Accuracy	0.9914	**0.9917**	0.9911
Precision	0.8916	**0.9589**	0.8387
Recall	0.5781	0.5469	0.6094
F2	0.6218	0.5983	0.6446
MCC	0.7142	0.7209	0.7107
RUSBoost			
Accuracy	0.9886	0.9916	0.9901
Precision	0.7157	0.8750	0.7037
Recall	0.5703	0.6016	**0.7422**
F2	0.5945	0.6417	**0.7342**
MCC	0.6332	**0.7217**	0.7176
NNET			
Accuracy	0.9882	0.9910	0.9876
Precision	0.7071	0.9559	0.6832
Recall	0.5469	0.5078	0.5391
F2	0.5728	0.5603	0.5628
MCC	0.6160	0.6933	0.6007

F2 scores during the cross validation stage. These results also mirror the algorithm's performance in the testing phase as it also scored the highest marks in these categories. using the complete dictionary as a preprocessing tool for AdaBoost allowed for AdaBoost to score the highest accuracy and precision in the cross validation results, as well as, the testing results. The highest marks in each metric mirrored each algorithms performance in the testing results, as each algorithm was able to perform the best in the same metrics no matter the phase of the project.

Results differ between the cross validation scores and the testing scores. The cross validation scores are lower across the board for the ensemble methods. The neural network scores for the complete and overcomplete dictionary cases scored similarly between the two testing phases. The metrics do not differ too much and the precision score for the complete dictionary case is higher in the cross validation case. The recall scores for every variant of each algorithm scored lower in the cross validation set when compared to the results on the testing set of data. These differing results come from the data that the algorithm uses. When using the cross validation data the algorithms have already seen this data before and we make decisions on how to tune the hyperparameters for the models based off of the algorithms performance. This data is biased since models have used this data at one point in the overall training process as a data point. Its performance on a data point that is has seen before will therefore bias the results of the model. The testing dataset is used to find the true unbiased performance of the final model. This dataset cannot be used to make hyperparameter decisions as it will then bias our results and ruin the testing dataset. Once the proper hyperparameters were tuned using the biased results of the cross validation stage we were able to see the models performance on an unbiased dataset. This will show how the models adapt to the real world and we can find the true error rate of our classifiers. The classifiers performed better once the algorithms were able to see the unbiased testing data.

The final results and the cross validation results verify the effectiveness of using the

complete and overcomplete dictionary as a preprocessing technique for lidar data. In both testing incidences the complete dictionary in conjunction with RUSBoost found that this ensemble method combined with this particular preprocessing dictionary found the best overall performance of a classifier. The overcomplete dictionary with RUSBoost scored the highest recall rate and allowed for the highest rate of insect detection in each phase of the research. These results match between the two differing results and help verify that sparse coding is an effective tool at preprocessing lidar data.

Table 4.3: Final AdaBoost Hyperparameter Values

Parameter	Value		
	D512	D1024	D2048
Learning Cycles	187	303	132
Learning Rate	.0407	.1414	.0513
Max Number of Splits	169510	138	13480
Min Leaf Size	11	613	65
Split Criterion	gdi	gdi	gdi
FN Cost	13	14	12
Undersampling ratio	.9000	0.3000	0.3000
# Synthetic Insects	750	750	250

The final hyperparameter values for AdaBoost are seen in Table 4.3. It can be seen that the full grid search found that the Gini's diversity index (gdi) was the best criterion to split the classes. The learning cycles, max number of splits, and leaf sizes very between algorithms. The undersampling ratios for the overcomplete and complete dictionaries were identical. This ratio may have aided these algorithms in the their classifications and helped

them score well in the metrics. The undercomplete and complete dictionary's high number of synthetic insects may have inversely aided these algorithms in their performance of classifying non-insect images, since their recall scores are lower than the overcomplete dictionary results which used only 250 synthetic insects. The higher false negative cost score for the complete dictionary, penalized misclassifications the most from the three versions of the algorithm. This higher misclassification cost may have increased its performance by improving its ability to identify images that more than likely do not look like they contain an insect. This may have aided it in scoring the highest precision score, as the algorithm learned what non-insect images looked like and was able to classify them much easier than images that contained insects. This also may attribute to its lower recall score, as images that contain an insect near the hard target may have caused AdaBoost to mislabel them. The FN cost may have had an affect on the classifiers and their performances, as the classifier with the highest FN cost performed the best overall. The remaining hyperparameters may have aided this FN cost, however with such varying hyperparameters between the three variants the parameters that allowed the complete dictionary version to perform the best remain unclear.

The tuned hyperparameters for RUSBoost are highlighted in Table 4.4. The exhaustive grid search found that the number of synthetic insects required for RUSBoost was much lower than the the other algorithms. AdaBoost had two versions with 750 synthetic insects and the neural network had two versions with 1000+ synthetic insects. The number of synthetic insects may not affect the performance of the algorithms by very much. The FN cost for the complete and overcomplete dictionary versions were the highest of the ensemble methods. The number of learning cycles for the three variants were on the higher end for the ensemble methods, with all three having a higher number of iterations when compared to AdaBoost. The undercomplete variant had 395 iterations, while the complete and overcomplete had 493 and 490 learning cycles respectively. The higher number of learning cycles may have aided these two variants in obtaining their high scores. Learning cycles allow each algorithm

Table 4.4: Final RUSBoost Hyperparameter Values

Parameter	Value		
	D512	D1024	D2048
Learning Cycles	395	493	490
Learning Rate	.7991	.0503	.6168
Max Number of Splits	35	144203	17
Min Leaf Size	15	3	2
Split Criterion	gdi	gdi	deviance
FN Cost	2	15	20
Undersampling ratio	0.1500	0.6000	0.9000
# Synthetic Insects	250	250	0

to see the data multiple times and allow it to optimize the parameters for the classifier. The learning rates, maximum number of splits, and undersampling ratios varied between each of the algorithms. The split criterion was also a varied between the algorithms with the overcomplete variant using a deviance split criterion versus the gdi criterion the other versions used. The minimum leaf size was on the lower end for the complete and overcomplete versions with the algorithms only using 3 and 2 leaf sizes respectively. The undersampling ratios were also varied among the three, with complete and overcomplete versions using the highest ratios among the ensemble methods. The combination of these hyperparameters for each of the algorithms as unique to each algorithm. By using an exhaustive grid search by evaluating the MCC score, each hyperparameter was chosen because it was able to optimize this MCC score. For RUSBoost the higher undersampling ratios, lower number of synthetics insect, high number of learning cycles, and the high FN cost were able to be used by each

algorithm to produce exceptional results.

Table 4.5: Final Neural Network Hyperparameter Values

Parameter	Value		
	D512	D1024	D2048
Layer Size	10	51	13
Lambda	3.0953e-08	8.8358e-06	9.0011e-12
Activation	relu	tanh	relu
Undersampling Ratio	0.6000	0.6000	0.3000
# Synthetic Insects	1000	0	1250

Figure 4.5, shows the final tuned hyperparameters for the neural network algorithm. The neural network in this research is a single hidden layer network with varying numbers of nodes. The layer size defines the number of nodes in the hidden layer of the neural network. The number of nodes varies among the three variants with the undercomplete and overcomplete versions of the neural network only containing 10 and 13 nodes. The neural network that utilized the complete dictionary had 51 nodes in its hidden layer. The addition of extra nodes did not increase the overall performance of the algorithm. The neural network that used the least amount of nodes, the undercomplete variant with 10 nodes, actually performed the best out of the neural networks. However the "best" performance still had a lower MCC score than the previous AdaBoost results, as well as the previous neural network results. The nonlinear activation functions used by each neural network were either the rectified linear unit (relu) function or the hyperbolic tangent (tanh) function. The relu function was found to work the best with the undercomplete and overcomplete dictionaries, whereas the complete dictionary version found the tanh function to give the

best results. The lambda parameter, otherwise known as the regularization term strength, varied between the three versions with the complete dictionary having the largest strength. The undersampling ratios were either 0.6000 or 0.3000. These ratios did not appear to have any large impact on the results as the undercomplete and complete variations used the same ratio with differing MCC scores. The number of synthetic insects were also observed to be different for each classifier as the complete dictionary neural network used no synthetic insects and the undercomplete and overcomplete networks used 1000 and 1250. However, these synthetic insect containing versions had varying performances overall with the undercomplete version scoring an MCC score of .7381 and the overcomplete scoring a .6543. The varying hyperparameters for each neural network show no clear indication on what may or may not have affected the effectiveness of each classifier.

The cost weighting hyperparameter may be the most important tool for the ensemble methods. In each of the algorithms the versions with the highest FN cost were capable of scoring the highest in each of the metric categories. The AdaBoost model using the complete dictionary had the highest FN cost of its three variants. This version was able to perform the best of the three, boasting the highest MCC score of the variants, as well as, scoring the highest accuracy and precision scores in the research. The RUSBoost models using the complete and overcomplete dictionaries had higher FN cost scores than the AdaBoost's highest FN cost of 14. These two variants managed to score the best MCC, accuracy, recall, and F2 scores in the project. The cost weighting of misclassifications may have been a very important parameter for these algorithms and allowed them to learn parameters to classify the data in an efficient manner. Another important example of this was the undercomplete dictionary variant of RUSBoost. This variant had the lowest FN score of the ensemble methods and it correspondingly had the lowest MCC score, also interpreted as having the lowest overall performance. This algorithm also had the lowest undersampling ratio, meaning that there were many more examples of the majority class

when compared to the minority class. This may have also affected the performance of this algorithm. The undersampling ratios may have aided classifiers in detecting insects by somewhat balancing the dataset. The RUSBoost algorithm that was preprocessed with the overcomplete dictionary had an undersampling ratio of 90%. This correlates to 90% of the majority class (7216 insect containing images) being removed prior to training. The algorithm likely trained on approximately 721 $\pm$ a few non-insect images and 128 insect containing images. This closer to "balanced" dataset highlights the large class imbalance that this dataset imposes. After the training set was reduced by 10% during KSVD training, and then further reduced by undersampling the 7216 non-insect images, there are still many more examples of non-insect containing images.

Misclassified images can be seen in Figures 4.2 and 4.3. Image (a) highlights an image that was misclassified by every classifier. This image may have contained too much noise in the same row as the insect. It can be seen that in the same row as the insect the data on the left hand side of the image appears to also follow the same time limited aspects of an insect. This noise is lower in magnitude but may have introduced some misleading harmonics when the frequency domain features were extracted. This seemingly time limited noise may have also interfered with the time domain features and increased the standard deviation of the insect row and place it closer to non-insect images in the feature space. Image (b) was misclassified by the ensemble methods. This image contained noise throughout the image, especially near the top of the image. This concentrated noise at the top of the image came from a lidar observation where the hard target was closer to the system. This band of concentrated noise may have had some aspects that introduced some features that placed it closer to insect images. More examples of misclassified images are in Appendix C.

Vannoy et al., previously found that the AdaBoost algorithm performed the best out of the three algorithms. It was able to correctly classify 72.73% of images that contained insects and was able to reduce 98.14% of the images that are required for the entomologist to

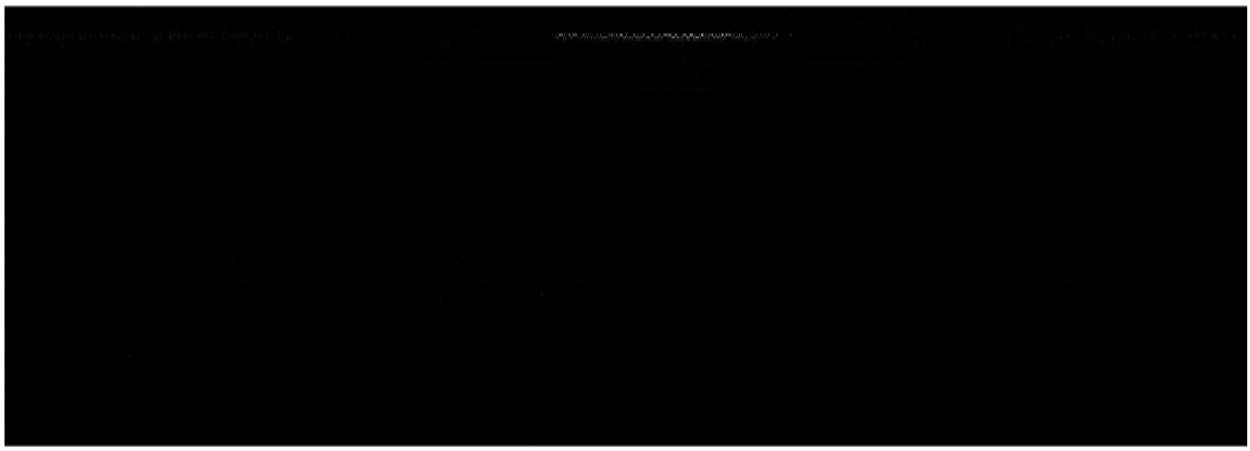

Figure 4.2: Image, preprocessed by the complete dictionary, that was misclassified by every algorithm.

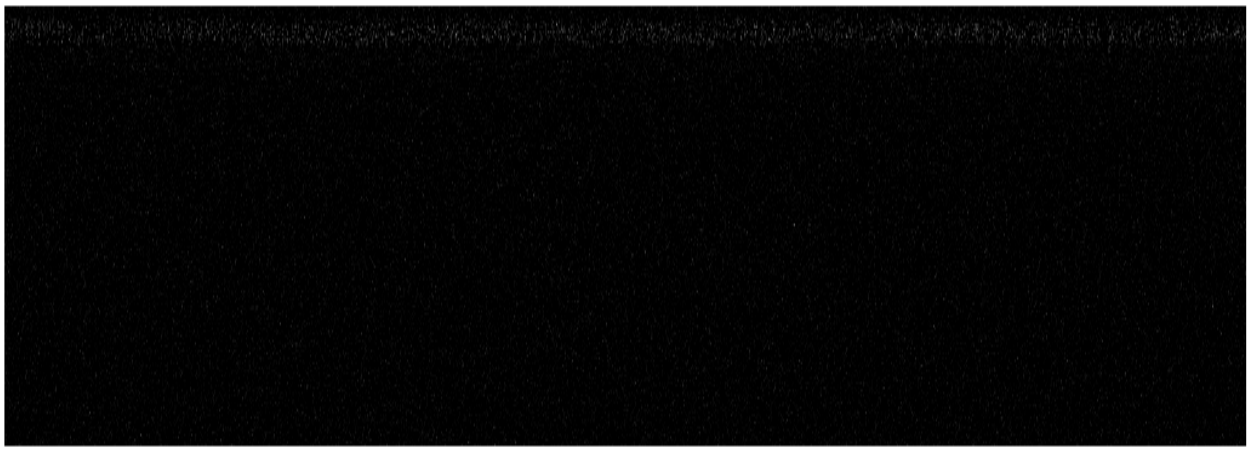

Figure 4.3: Image misclassified by RUSBoost and AdaBoost originally preprocessed with the complete dictionary

evaluate. The best results from this research can correctly classify 93.18% of insect containing images using the overcomplete dictionary with RUSBoost. Using the complete dictionary with RUSBoost the algorithm was able to maintain a non-insect image classification rate of 99.7% while increasing the classification of insect images to 84.09%.

AdaBoost boasts the best average MCC score of .8542 between all three variations.

The AdaBoost algorithm was consistently able to produce great results no matter the dimensions of the dictionary. AdaBoost obtained the next three highest MCC scores after the RUSBoost's high MCC score of .8682. The complete dictionary gave the best results in conjunction with AdaBoost. It boasted top scores in precision, and tied with the highest accuracy score. Using sparse coding as a preprocessing technique enable AdaBoost to have a 7.9% increase in overall performance, when comparing MCC scores. AdaBoost combined with the overcomplete dictionary scored the highest F2 and recall scores among the three dictionaries, enabling AdaBoost to classify the most insect images in the test data set. The overcomplete dictionary was able to detect 15.61% more insects than the previous AdaBoost results. The only score that did not increase with sparse coding was the precision score of the overcomplete dictionary. However, the precision score only decreased by .0079, from .8889 to .8810 or .89%.

AdaBoost using the undercomplete dictionary was able to retain 79.55% of insect containing images, eliminate 1895 out of 1942 images (97.57%), and correctly classify 99.78% of images containing no insects. The complete dictionary was able to retain 77.27% of insect containing images, eliminate 1899 out of 1942 images (97.78%), and correctly classify 99.94% of non-insect images. The overcomplete dictionary retained 84.09% of insect images, eliminated 1892 (97.42%) images, and classified 99.73% of non-insect images correctly. The previously found results retained 72.72% of images containing insects, eliminated 1898 (97.73%) of images in the test set, and correctly classified 99.78% of the non-insect images. Each of the AdaBoost variants show an increase in the retention of the insect containing images and an increase in the correct classification of non-insect images. This is true except for the overcomplete variant, where one more non-insect image was incorrectly classified and this percentage decreased. However, this version of AdaBoost boasts the second highest accuracy rate, meaning that it was able to correctly classify 1992 (99.38%) images. The average accuracy score among the three variants scored the highest of the

three algorithms at 99.38%. Utilizing sparse coding as a preprocessing technique has shown increased performance in insect detection while also maintaining a high accuracy rate overall.

RUSBoost had overall improvements across the three variants. The average MCC score for RUSBoost was 0.8179. The only scores that did not improve were the F2 and the recall scores of the undercomplete variant. Similar to the neural network the undercomplete dictionary in conjunction with RUSBoost caused an increase of the misclassification of insect images. The F2 score is a weighted combination of the precision and recall scores, giving more weight to the recall score. Therefore the lower F2 score can be be attributed to the lower recall score, since it carries more importance in the F2 score. The only other score that showed no improvement and maintained the same value is the recall score for the complete dictionary alternative. The difference between the RUSBoost's top score and the AdaBoost top MCC score was .0044. RUSBoost classified the same number of insect images while increasing the other scores. It was able to correctly classify more of the non-insect images which allows for the elimination of images that will need to be inspected. This will increase the accuracy, precision, F2, and MCC scores all while maintaining the same recall rate as the previous results. This is an important improvement as it shows the effectiveness of using sparse coding as a preprocessing technique. By training KSVD and utilizing the resulting dictionary properly, the detection rate of a minority class in lidar data can be increased. For this research, the complete dictionary's overall score of the algorithm increased the overall performance of RUSBoost by 22.87%. This correlates to the best MCC and accuracy score of the research. The complete dictionary with RUSBoost obtained the best results in the four confusion matrix categories.

RUSBoost preprocessed with the overcomplete dictionary had an overall increase of 17.35%, when comparing the MCC scores. It obtained the best recall and F2 scores in the project. This points to the highest number of correctly classified insect images. From the test set the overcomplete dictionary allowed RUSBoost to correctly classify 93.18% of the

insect images. AdaBoost with the overcomplete dictionary, RUSBoost with the complete dictionary, and RUSBoost from the previous results scored the second best recall rate of .8409. RUSBoost with the overcomplete dictionary was able to correctly classify 10.80% more insects. When comparing this to previous results best performing algorithm, AdaBoost, the recall rate was increased by 28.11%. This result means that when sparse coding is used as a preprocessing technique, the RUSBoost algorithm is able to classify 28.11% more images containing insects than previously found. However, this high recall came at a trade off. The algorithm had a lower precision score than the other variants of the dictionary. This indicates that the algorithm falsely labeled more non-insect images. As seen in matrix (c) from 4.1, there were 14 mislabelled non-insect images. The algorithm is likely placing more weight on identifying any image that appears to have an insect, rather than correctly identifying more of the majority class.

RUSBoost using the undercomplete dictionary was able to retain 65.91% of insect containing images. This algorithm was capable of eliminating 1901 (97.88%) of images in the testing dataset, and correctly classify 99.78% of non-insect containing images. The RUSBoost variant preprocessed by the complete dictionary was able to retain 84.09% of insect images. This variant was also capable of automatically removing 1893 (97.47%) images from the dataset and could correctly classify 99.78% of non-insect containing images. Finally, the RUSBoost version preprocessed by the overcomplete dictionary was able to retain, the research's best, 93.18% of insect containing images. This version automatically removed 1879 (96.75%) images from the dataset and correctly classified (99.25%) of non-insect images. The results verify that using sparse coding as a preprocessing technique has show improvements upon previous results and was able to classify 28.11% more insects than the previous research done in this field.

The neural network performed worse overall compared to previous results with an average MCC score of 0.6846. The results show little difference between the neural

networks preprocessed by the complete and overcomplete dictionaries. The neural network preprocessed by the undercomplete dictionary performed the best of the variants with a MCC score of .7381 when compared to the other MCC scores of .6616 and .6543 from the complete and overcomplete variants. This model is the only method that showed no improvements when in comparison to the previous results. The AdaBoost and RUSBoost models showed improvements in each score when using the varying dictionaries. The worst results for those ensemble models still managed to have a higher overall score when compared to the highest MCC score of the neural network. The results appear to have an inverse affect compared to the other models. If the dimensions of the dictionary increase the neural network is unable to classify insects better. This is an interesting result, since the other classifiers were able to increase detection rate with the complete and overcomplete dictionaries. Those models performed worse utilizing the undercomplete dictionary as the difference image likely had more noise in the resulting image. However the neural network in combination with the undercomplete dictionary was able to correctly classify 68.18% of insect images. This was an increase of 7.1% when compared to the previous results for the neural network. However, this is at the cost of incorrectly classifying more non-insect images. The previous precision was .9333, if compared to the precision value of .8108 this means an increase of incorrectly classified non-insect images.

A single layer neural network preprocessed by the undercomplete dictionary was able to retain 68.18% of insect containing images. This version was able to eliminate 1897 (97.68%)images and correctly classify 99.62% of non-insect images. The neural network variant using the complete dictionary was able to retain 52.27%, the research's worst, of insect containing images. This network automatically removed 1907 (98.19%) images from the dataset and correctly classified 99.78% of non-insect images. The neural network preprocessed by the overcomplete dictionary was only able to retain 54.55% of insect containing images. This version of the neural network was able to remove 1904 (98.04%)

images from the testing dataset, and correctly classify 99.68% on non-insect images. The results for the neural network were not improved by using sparse coding as a preprocessing technique.

The new research suggests that the neural network does not perform as well when analyzing sparse coding reconstruction images with the preprocessed data and its results were worse overall. With the undercomplete dictionary the AdaBoost outperformed most of the previous results and was able to classify more images containing insects than the previously used AdaBoost. The RUSBoost algorithm performed the best out of the algorithms used. Using the complete dictionary, RUSBoost was able to achieve the highest MCC and Accuracy scores. The previous research used the MCC score as the main component to analyze how well the algorithm performed. The MCC score increased to 0.8682, compared to the previous best score of 0.8000. The recall score is another important metric to consider when analyzing the performance of the algorithms. It is important to have a high recall rate since the value of identifying insects is of high significance. The RUSBoost using the overcomplete dictionary achieved this metric and comparatively had the best F2 score. Ensemble methods appear to be able to perform well in the environment produced by sparse coding, whereas the single layer neural network did not perform well in the feature space.

CONCLUSION

In this research the ensemble methods, AdaBoost and RUSBoost, scored the four highest MCCs and each had high scores in every score category. The neural network, performed worse overall when compared to previous results. RUSBoost scored the highest MCC and increased the insect detection rate by 28.11% and achieve a recall score of .9318. In our test set this correlates to correctly classifying 41 out of 44 images containing insects. AdaBoost, with a complete dictionary, had the highest precision score at .9714 and the highest accuracy at .9943. These scores show the algorithms good performance in correctly classifying 1889/1890 of images containing no insects. In this work the recall score is weighted more than the precision score due to a goal of the project to find all images containing insects. Having a low precision score is not a major issue in this work, as long as the recall rate is a sufficiently high. The goal is to classify any image that contains an insect, if some non-insect images get incorrectly classified they can be easily spotted by an expert. Eliminating easily classifiable images was one of the main goals of the project, as we want to use entomologists time more efficiently. Using this metric the project was successfully able to eliminate 1879 out of 1942 images (96.75%) of images and require only manual inspection of 55 images. The project was able to retain 93.18% of insect containing images. The results of this project will allow entomologists to utilize their time effectively by eliminating the amount of time spent going through each individual lidar image searching for an insect. Instead of spending hours searching through each of the 1942 images, the entomologist can instead look through only 63 images that will help sample an insect population. This was only possible through the use of sparse coding as a preprocessing technique, along with the RUSBoost and AdaBoost ensemble learners.

Having a dictionary with many atoms is beneficial to reconstructing the lidar images. Each lidar image contains a hard target, its reflection, empty space, or an insect. The more

atoms that can be optimized to reconstruct the hard target and its reflection, the better the feature extraction and classifying models are able to perform. Each observation in a lidar image contains noise; this noise will creep into other range boxes that would normally contain empty space. Since every observation contains some noise, this means that the hard target in every image will look slightly different and could look different from image to image. Having a complete, or an overcomplete, dictionary helps to alleviate this issue. The more atoms that are available to choose from means that these slight differences can hopefully be modeled. Modeling the hard target and its reflection is vital to generating the difference images. If these can both be eliminated in the difference image, then the only differences that can make it to the final signal are noise and insects. Having the ability to model the variances on the hard target allows for the use of only four nonzero coefficients to reconstruct all images. This allows for the high dimensional portions of the image to stand out in the difference image. Since the feature extraction algorithm uses these images as inputs, it allows it to further separate the insects from the noise in the feature space. This feature space is where the machine learning models are then trained and are able to learn the parameters to classify the data.

The results verify the benefit of using sparse coding as a technique to process lidar data. Training KSVD on non-insect data allows the dictionary to learn features from the majority class in the data. This allows for the sparse vector to reconstruct a relevant signal from basis functions in the learned dictionary. Due to the low dimensionality of lidar data, sparse coding can reproduce non-insect lidar images, without noise, using only 4 features from a learned dictionary. Sparse coding struggles to reconstruct the high dimensional portions of lidar data. These portions of the data include gaussian noise and the modulated signal from the insect and its wings. The learned dictionary does not contain the appropriate learned features to reconstruct the insect signal. Thus, analyzing the sparse coding difference images is conducive to detecting insects, outperforming machine learning methods applied on the

original input images.

Future Work

Future work includes the testing of a linear classifier, a deep neural network, and using a convolutional neural network using the preprocessed data. Using the difference images may increase the separation of the insect data from non-insect data in the feature space. The use of a linear classifier such as a support vector machine may prove to be beneficial for detecting insects. Having a deep/multilayer neural network may provide another option that could boost results. A single layer neural network performed poorly using the preprocessed images, so the addition of multiple layers may increase the algorithms performance. With the added depth of the neural networks more complex relationships can be found through non-linear mappings and allow for improved insect detection. Using a convolutional neural network with the processed images should increase the insect detection rate as they are typically used with images. The CNN should take in the difference images created by this work since most of the image should contain noise, except for the insects that make it through to the difference image. The insect's higher magnitude and distinct waveform can be used in a CNN as they should be able to find these portions of the image. CNNs are also known to have translational invariance, this will be effective in finding insects that appear in differing places between images.

REFERENCES CITED

[1] M. Aharon, M. Elad, and A. Bruckstein. "rmK-SVD: An Algorithm for Designing Overcomplete Dictionaries for Sparse Representation". In: *IEEE Transactions on Signal Processing* 54.11 (Nov. 2006), pp. 4311–4322. DOI: 10.1109/tsp.2006.881199.

[2] Abdullah Emin Akay, Hakan Oğuz, Ismail Rakip Karas, and Kazuhiro Aruga. "Using LiDAR technology in forestry activities". In: *Environmental Monitoring and Assessment* 151.1-4 (Mar. 2008), pp. 117–125. DOI: 10.1007/s10661-008-0254-1.

[3] M.L. Buffington and R.A. Redak. "A comparison of vacuum sampling versus sweep-netting forarthropod biodiversity measurements in California coastal sage scrub". In: *Journal of Insect Conservation* 2.2 (1998), pp. 99–106. DOI: 10.1023/a:1009653021706.

[4] T. Tony Cai and Lie Wang. "Orthogonal Matching Pursuit for Sparse Signal Recovery With Noise". In: *IEEE Transactions on Information Theory* 57.7 (July 2011), pp. 4680–4688. DOI: 10.1109/tit.2011.2146090.

[5] Brandon T. Carroll, Bradley M. Whitaker, Wayne Dayley, and David V. Anderson. "Outlier Learning via Augmented Frozen Dictionaries". In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 25.6 (June 2017), pp. 1207–1215. DOI: 10.1109/taslp.2017.2690567.

[6] Corinna Cortes and Vladimir Vapnik. "Support-Vector Networks". In: *Machine Learning* 20.3 (1995), pp. 273–297. DOI: 10.1023/a:1022627411411.

[7] Ralph O. Dubayah and Jason B. Drake. "Lidar Remote Sensing for Forestry". In: *Journal of Forestry* 98.6 (June 2000), pp. 44–46. ISSN: 0022-1201. DOI: 10.1093/jof/98.6.44. eprint: https://academic.oup.com/jof/article-pdf/98/6/44/22558157/jof0044.pdf. URL: https://doi.org/10.1093/jof/98.6.44.

[8] Michael Elad and Michal Aharon. "Image Denoising Via Sparse and Redundant Representations Over Learned Dictionaries". In: *IEEE Transactions on Image Processing* 15.12 (Dec. 2006), pp. 3736–3745. DOI: 10.1109/tip.2006.881969.

[9] E. Fishler, A. Haimovich, R.S. Blum, L.J. Cimini, D. Chizhik, and R.A. Valenzuela. "Spatial Diversity in Radars—Models and Detection Performance". In: *IEEE Transactions on Signal Processing* 54.3 (Mar. 2006), pp. 823–838. DOI: 10.1109/tsp.2005.862813.

[10] Y. Freund and R. Schapire. "Experiments with a New Boosting Algorithm". In: *International Conference on Machine Learning* (1996). URL: https://www.semanticscholar.org/paper/68c1bfe375dde46777fe1ac8f3636fb651e3f0f8.

[11] Yoav Freund and Robert E Schapire. "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting". In: *Journal of Computer and System Sciences* 55.1 (Aug. 1997), pp. 119–139. DOI: 10.1006/jcss.1997.1504.

[12] Zhi Gao and Hong Ji. "Transform Learning Based Sparse Coding for LiDAR Data Denoising". In: *IEEE Signal Processing Letters* 26.3 (Mar. 2019), pp. 480–484. DOI: 10.1109/lsp.2019.2895974.

[13] Zhi Gao, Qingquan Li, Ruifang Zhai, Mo Shan, and Feng Lin. "Adaptive and Robust Sparse Coding for Laser Range Data Denoising and Inpainting". In: *IEEE Transactions on Circuits and Systems for Video Technology* 26.12 (Dec. 2016), pp. 2165–2175. DOI: 10.1109/tcsvt.2015.2492859.

[14] F. Stuart Chapin III, Erika S. Zavaleta, Valerie T. Eviner, Rosamond L. Naylor, Peter M. Vitousek, Heather L. Reynolds, David U. Hooper, Sandra Lavorel, Osvaldo E. Sala, Sarah E. Hobbie, Michelle C. Mack, and Sandra Dıaz. "Consequences of changing biodiversity". In: *Nature* 405.6783 (May 2000), pp. 234–242. DOI: 10.1038/35012241.

[15] Sooyoung Kim, Robert J. McGaughey, Hans-Erik Andersen, and Gerard Schreuder. "Tree species differentiation using intensity data derived from leaf-on and leaf-off airborne laser scanner data". In: *Remote Sensing of Environment* 113.8 (Aug. 2009), pp. 1575–1586. DOI: 10.1016/j.rse.2009.03.017.

[16] M. Mahmoudi and G. Sapiro. "Sparse Representations for Range Data Restoration". In: *IEEE Transactions on Image Processing* 21.5 (May 2012), pp. 2909–2915. DOI: 10.1109/tip.2012.2185940.

[17] S.G. Mallat and Zhifeng Zhang. "Matching pursuits with time-frequency dictionaries". In: *IEEE Transactions on Signal Processing* 41.12 (1993), pp. 3397–3415. DOI: 10.1109/78.258082.

[18] C. R. Margules and R. L. Pressey. "Systematic conservation planning". In: *Nature* 405.6783 (May 2000), pp. 243–253. DOI: 10.1038/35012251.

[19] Ajmal Mian and Richard Hartley. "Hyperspectral video restoration using optical flow and sparse coding". In: *Optics Express* 20.10 (Apr. 2012), p. 10658. DOI: 10.1364/oe.20.010658.

[20] Jörg Müller and Roland Brandl. "Assessing biodiversity by remote sensing in mountainous terrain: the potential of LiDAR to predict forest beetle assemblages". In: *Journal of Applied Ecology* 46.4 (Aug. 2009), pp. 897–905. DOI: 10.1111/j.1365-2664.2009.01677.x.

[21] Camille Parmesan. "Ecological and Evolutionary Responses to Recent Climate Change". In: *Annual Review of Ecology, Evolution, and Systematics* 37 (2006), pp. 637–669. ISSN: 1543592X, 15452069. URL: http://www.jstor.org/stable/30033846.

[22] Peter W Price. *Insect ecology*. John Wiley & Sons, 1997.

[23] Thinal Raj, Fazida Hanim Hashim, Aqilah Baseri Huddin, Mohd Faisal Ibrahim, and Aini Hussain. "A Survey on LiDAR Scanning Mechanisms". In: *Electronics* 9.5 (Apr. 2020), p. 741. DOI: 10.3390/electronics9050741.

[24] Maxime Réjou-Méchain, Blaise Tymen, Lilian Blanc, Sophie Fauset, Ted R. Feldpausch, Abel Monteagudo, Oliver L. Phillips, Hélène Richard, and Jérôme Chave. "Using repeated small-footprint LiDAR acquisitions to infer spatial and temporal variations of a high-biomass Neotropical forest". In: *Remote Sensing of Environment* 169 (Nov. 2015), pp. 93–101. DOI: 10.1016/j.rse.2015.08.001.

[25] F. K. Rengers, G. E. Tucker, J. A. Moody, and B. A. Ebel. "Illuminating wildfire erosion and deposition patterns with repeat terrestrial lidar". In: *Journal of Geophysical Research: Earth Surface* 121.3 (Mar. 2016), pp. 588–608. DOI: 10.1002/2015jf003600.

[26] Kevin S Repasky, Joseph A Shaw, Ryan Scheppele, Christopher Melton, John L Carsten, and Lee H Spangler. "Optical detection of honeybees by use of wing-beat modulation of scattered laser light for locating explosives and land mines". In: *Applied optics* 45.8 (2006), pp. 1839–1843.

[27] F. Rosenblatt. "The perceptron: A probabilistic model for information storage and organization in the brain." In: *Psychological Review* 65.6 (1958), pp. 386–408. DOI: 10.1037/h0042519.

[28] W. G. Rudd and R. L. Jensen. "Sweep Net and Ground Cloth Sampling for Insects in Soybeans". In: *Journal of Economic Entomology* 70.3 (June 1977), pp. 301–304. DOI: 10.1093/jee/70.3.301.

[29] Chris Seiffert, Taghi M. Khoshgoftaar, Jason Van Hulse, and Amri Napolitano. "RUSBoost: A Hybrid Approach to Alleviating Class Imbalance". In: *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* 40.1 (Jan. 2010), pp. 185–197. DOI: 10.1109/tsmca.2009.2029559.

[30] Joseph A Shaw, Kevin S Repasky, John L Carlsten, Lee H Spangler, and David S Hoffman. *Optical detection of oscillating targets using modulation of scattered laser light*. US Patent 7,511,624. Mar. 2009.

[31] Joseph A Shaw, Nathan L Seldomridge, Dustin L Dunkle, Paul W Nugent, Lee H Spangler, Jerry J Bromenshenk, Colin B Henderson, James H Churnside, and James J Wilson. "Polarization lidar measurements of honey bees in flight for locating land mines". In: *Optics express* 13.15 (2005), pp. 5853–5863.

[32] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. *Practical Bayesian Optimization of Machine Learning Algorithms*. 2012. DOI: 10.48550/ARXIV.1206.2944.

[33] G. W. Stewart. "On the Early History of the Singular Value Decomposition". In: *SIAM Review* 35.4 (Dec. 1993), pp. 551–566. DOI: 10.1137/1035134.

[34] Gilbert Strang. "The Discrete Cosine Transform". In: *SIAM Review* 41.1 (Jan. 1999), pp. 135–147. DOI: 10.1137/s0036144598336745.

[35] J. A. Thomas, M. G. Telfer, D. B. Roy, C. D. Preston, J. J. D. Greenwood, J. Asher, R. Fox, R. T. Clarke, and J. H. Lawton. "Comparative Losses of British Butterflies, Birds, and Plants and the Global Extinction Crisis". In: *Science* 303.5665 (Mar. 2004), pp. 1879–1881. DOI: 10.1126/science.1095046.

[36] Joel A. Tropp and Anna C. Gilbert. "Signal Recovery From Random Measurements Via Orthogonal Matching Pursuit". In: *IEEE Transactions on Information Theory* 53.12 (Dec. 2007), pp. 4655–4666. DOI: 10.1109/tit.2007.909108.

[37] Trevor C. Vannoy, Kyle R. Rust, Joseph N. Aist, Riley D. Logan, Elizabeth M. Rehbein, Joseph A. Shaw, and Bradley M. Whitaker. "Automated Detection of Insects in Lidar Data". In: *2021 IEEE Research and Applications of Photonics in Defense Conference (RAPID)*. IEEE, Aug. 2021. DOI: 10.1109/rapid51799.2021.9521465.

[38] Trevor C. Vannoy, Trey P. Scofield, Riley D. Logan, Elizabeth M. Rehbein, Joseph A. Shaw, and Bradley M. Whitaker. *Dataset for Insect Lidar Supervised Classification*. Version v1.0.0. Zenodo, Sept. 2021. DOI: 10.5281/zenodo.5504411. URL: https://doi.org/10.5281/zenodo.5504411.

[39] Trevor C. Vannoy, Trey P. Scofield, Riley D. Logan, Elizabeth M. Rehbein, Joseph A. Shaw, and Bradley M. Whitaker. *Insect Lidar Supervised Classification*. Version 1.0.0. If you use this software, please cite it as below. Sept. 2021. DOI: 10.5281/zenodo.5504409. URL: https://doi.org/10.5281/zenodo.5504409.

[40] Trevor C. Vannoy, Trey P. Scofield, Joseph A. Shaw, Riley D. Logan, Bradley M. Whitaker, and Elizabeth M. Rehbein. "Detection of Insects in Class-Imbalanced Lidar Field Measurements". In: *2021 IEEE 31st International Workshop on Machine Learning for Signal Processing (MLSP)*. 2021, pp. 1–6. DOI: 10.1109/MLSP52302.2021.9596143.

[41] Ruisheng Wang, Jiju Peethambaran, and Dong Chen. "LiDAR Point Clouds to 3-D Urban Models: A Review". In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 11.2 (Feb. 2018), pp. 606–627. DOI: 10.1109/jstars.2017.2781132.

[42] B. A. Woodcock. "Pitfall Trapping in Ecological Studies". In: *Insect Sampling in Forest Ecosystems*. Blackwell Science Ltd, 2007, pp. 37–57. DOI: 10.1002/9780470750513.ch3.

[43] Qun Zhang, Ying Luo, and Yong-an Chen. "Micro-Doppler Feature Analysis and Extraction". In: *Micro-Doppler Characteristics of Radar Targets*. Elsevier, 2017, pp. 87–170. DOI: 10.1016/b978-0-12-809861-5.00005-9.

APPENDICES

APPENDIX A

CONFUSION MATRICES

The following confusion matrices represent the results of the classifiers on the test data set. The results for the image classification are shown for each classifier. Three dictionaries were used with each of the classifiers. The undercomplete, complete, and overcomplete dictionaries combined with each classifier are shown. The first confusion matrix represents the previous results best performing classifier. Then the AdaBoost, RUSBoost, and neural network results for this work are shown.

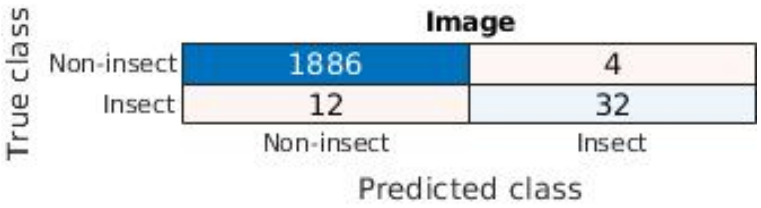

Figure A.1: Previous AdaBoost Confusion Matrix [40]

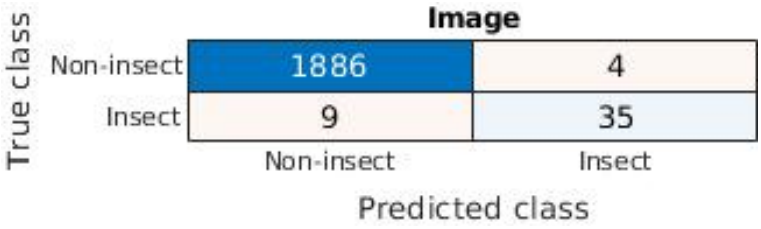

Figure A.2: AdaBoost Confusion Matrix Using the Undercomplete (D512) Dictionary

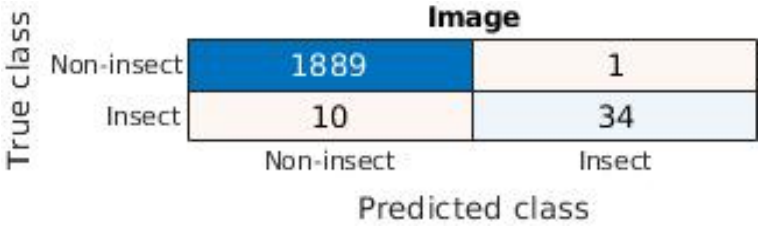

Figure A.3: AdaBoost Confusion Matrix Using the Complete (D1024) Dictionary

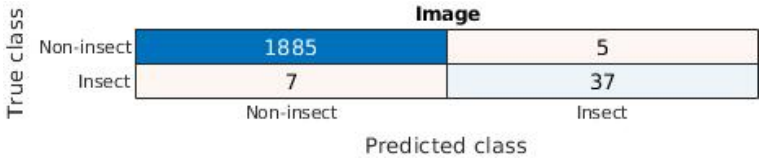

Figure A.4: AdaBoost Confusion Matrix Using the Overcomplete (D2048) Dictionary

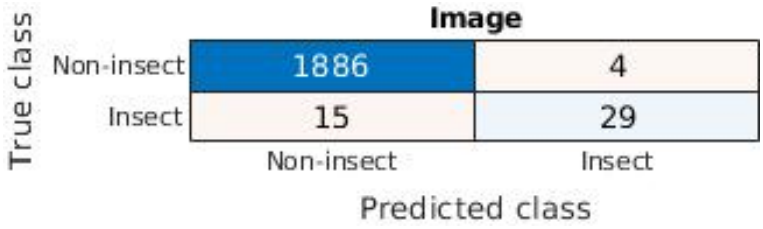

Figure A.5: RUSBoost Confusion Matrix Using the Undercomplete (D512) Dictionary

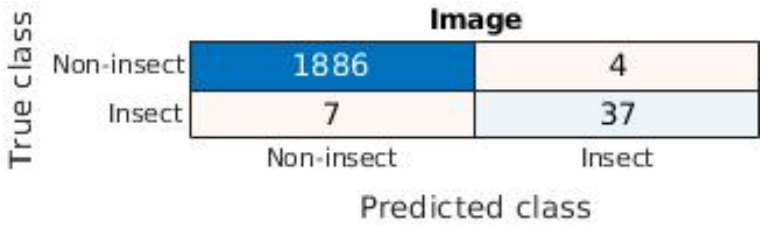

Figure A.6: RUSBoost Confusion Matrix Using the Complete (D1024) Dictionary

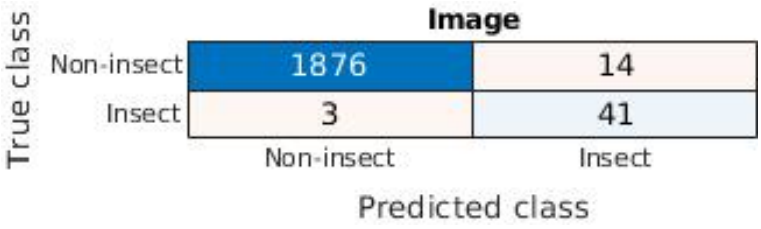

Figure A.7: RUSBoost Confusion Matrix Using the Overcomplete (D2048) Dictionary

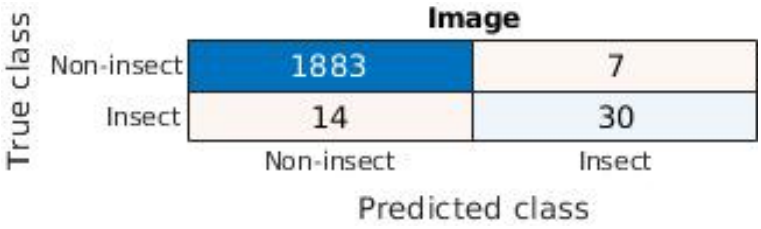

Figure A.8: Neural Network Confusion Matrix Using the Undercomplete (D512) Dictionary

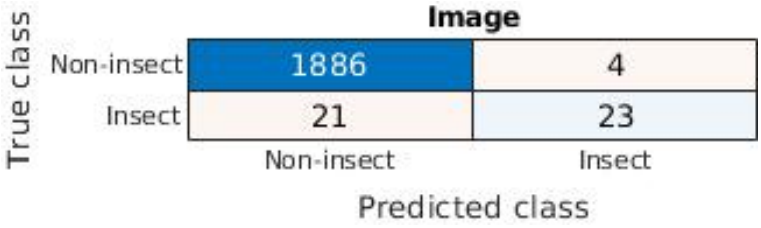

Figure A.9: Neural Network Confusion Matrix Using the Complete (D1024) Dictionary

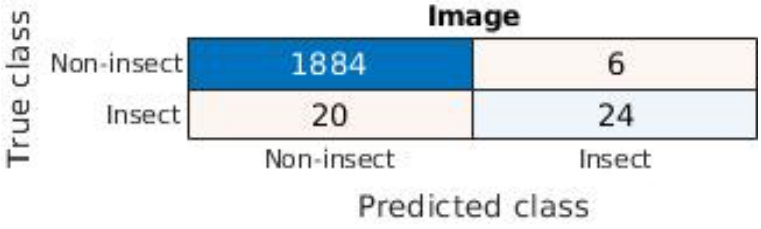

Figure A.10: Neural Network Confusion Matrix Using the Overcomplete (D2048) Dictionary

APPENDIX B

EXAMPLE IMAGES

The following images were selected randomly from the respective classes in the difference image dataset. These example images are to show that the signals in each lidar image vary. The insect's position and signal vary between images, and the hard targets also vary in position and signal between images. Some images contain a lot of noise whereas some images have a relatively low noise observation. Five images from each class will be highlighted here. The insect images will precede the non-insect images.

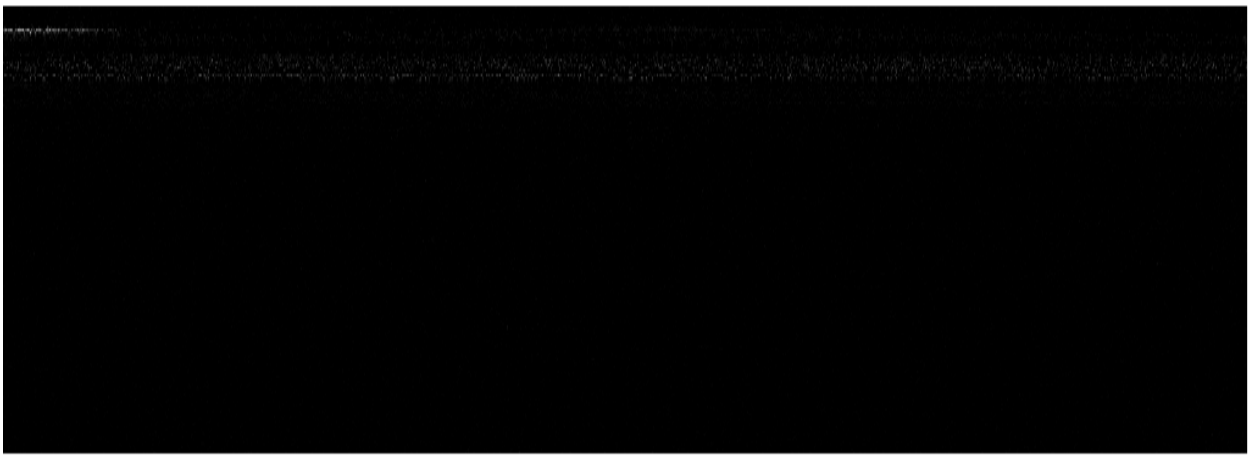

Figure B.1: Example of a difference image that contains an insect.

Figure B.2: Another example of a difference image that contains an insect.

Figure B.3: Example of a difference image that contains an insect.

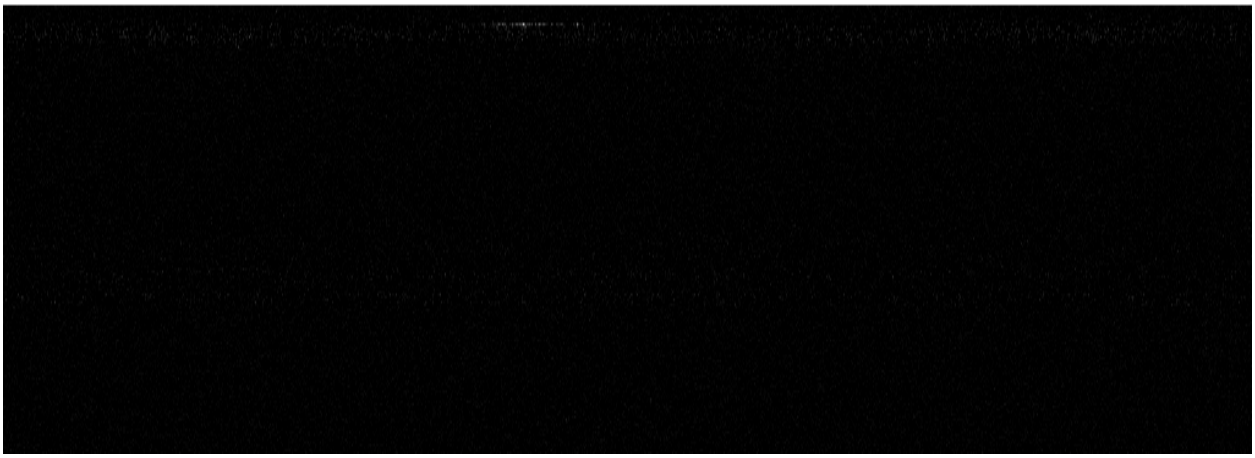

Figure B.4: Another example of a difference image that contains an insect.

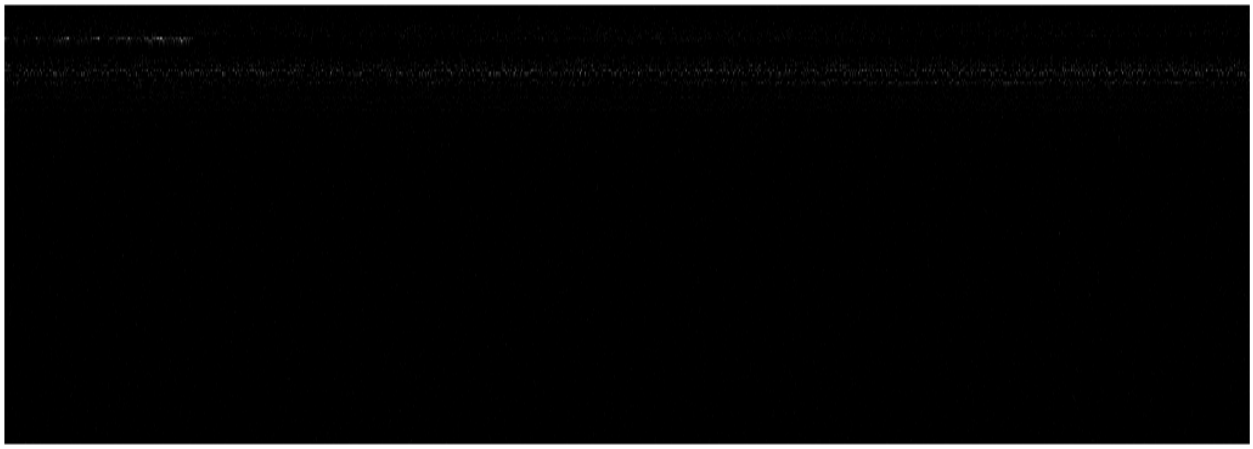

Figure B.5: Another example of a difference image that contains an insect.

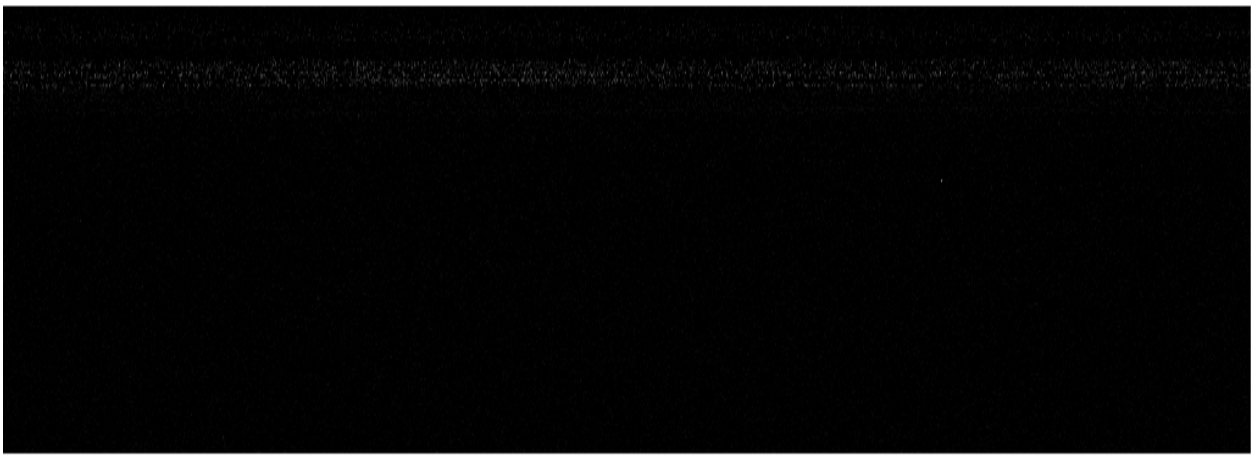

Figure B.6: Example of a difference image that does not contain an insect.

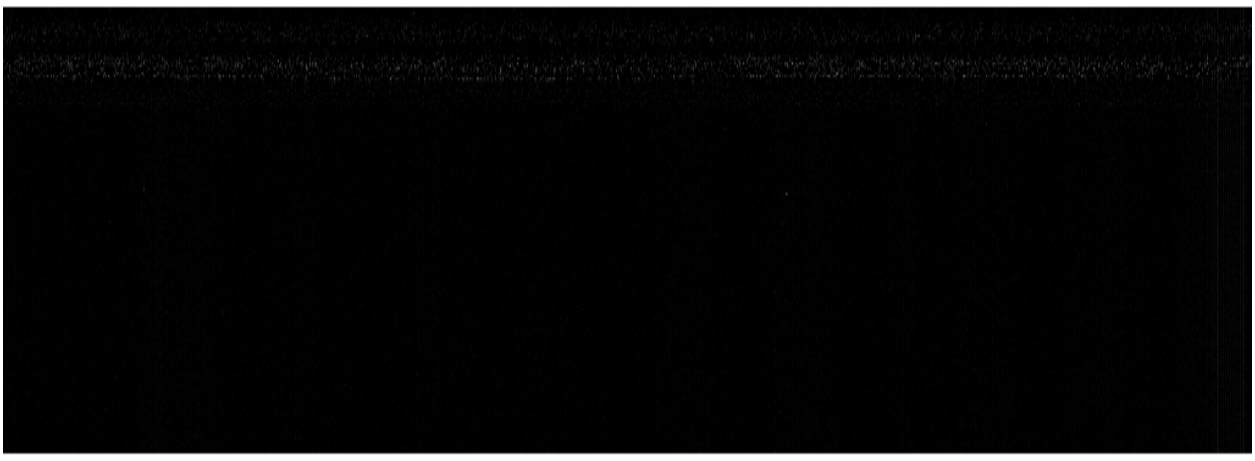

Figure B.7: Another example of a difference image that does not contain an insect.

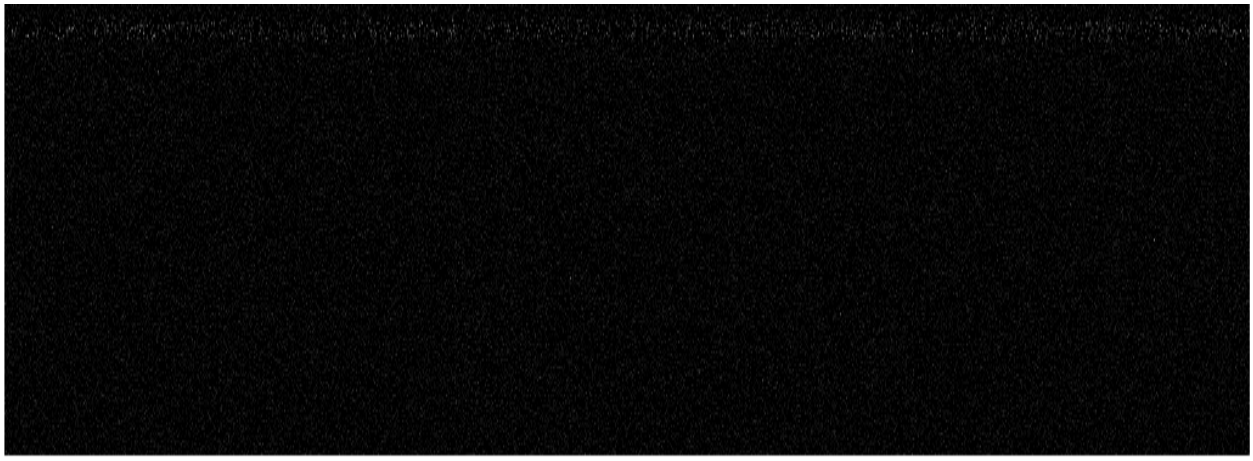

Figure B.8: Another example of a difference image that does not contain an insect.

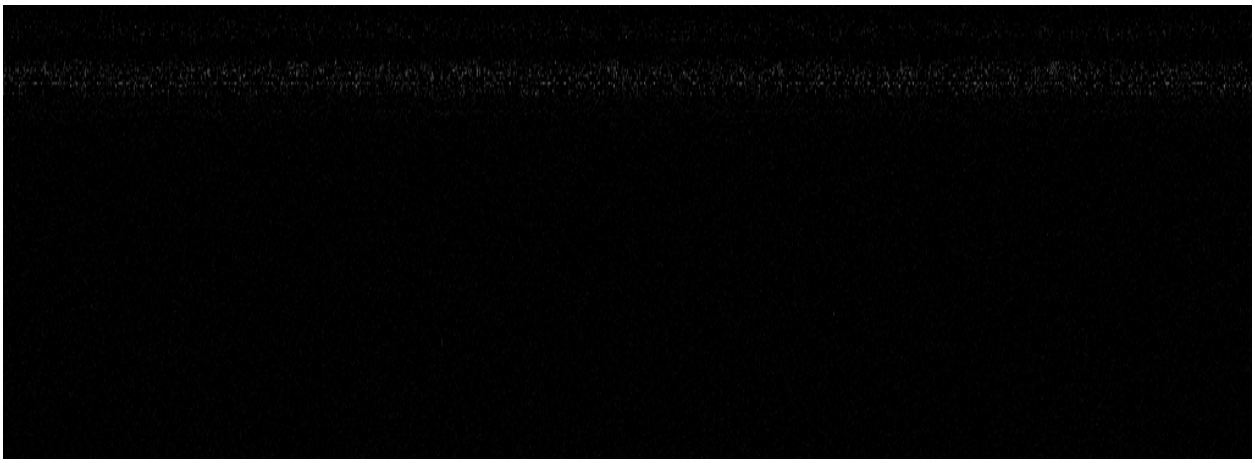

Figure B.9: Another example of a difference image that does not contain an insect.

Figure B.10: Another example of a difference image that does not contain an insect.

APPENDIX C

MISCLASSIFIED IMAGES

The following images are examples of misclassified images from the testing dataset. The difference images that were used in the pipeline will be shown here. Some of these images were misclassified by all algorithms and the rest of the images were either misclassified by two of the methods or just one method. These misclassified images come from both classes as some insect images were incorrectly classified and some non-insect images were misclassified. Ten of these misclassified images will be presented.

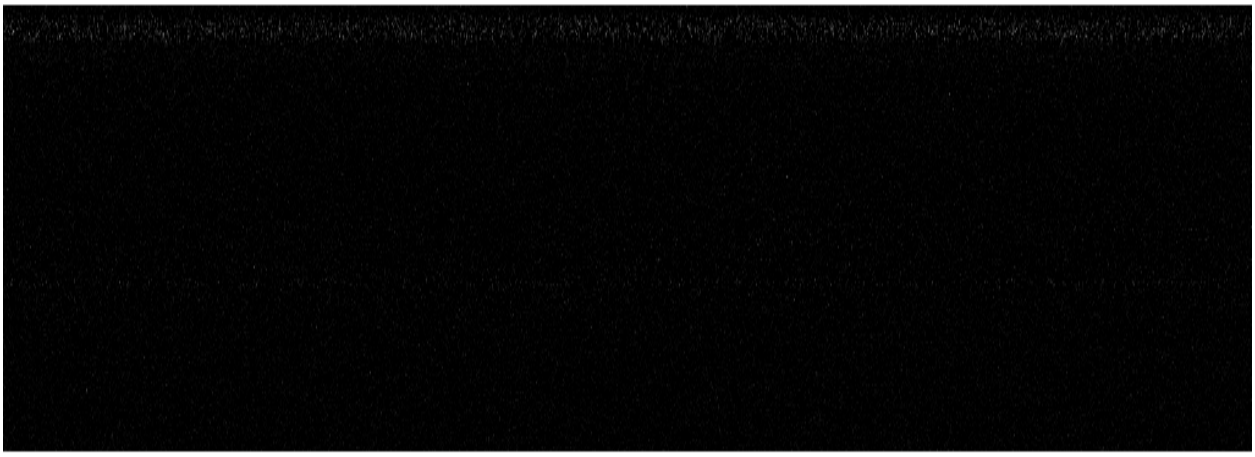

Figure C.1: Example of a difference image that was misclassified by AdaBoost.

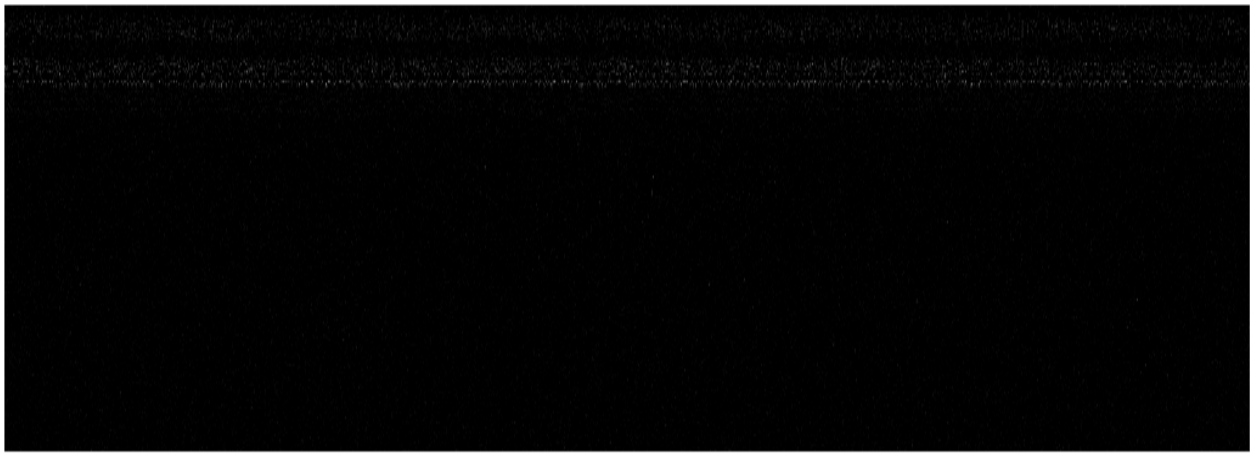

Figure C.2: Example of an image misclassified by the Neural Network

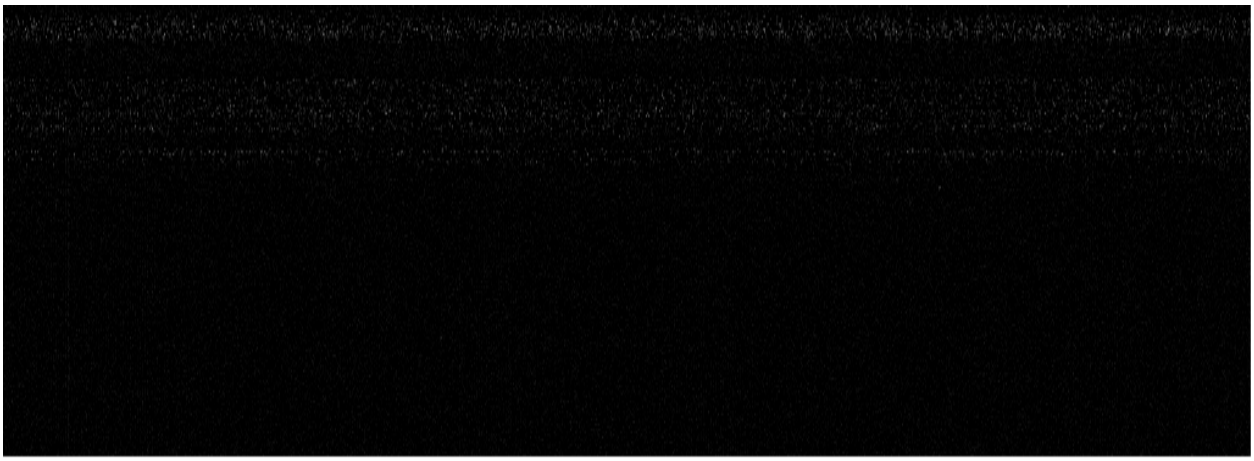

Figure C.3: Example of an image misclassified by RUSBoost.

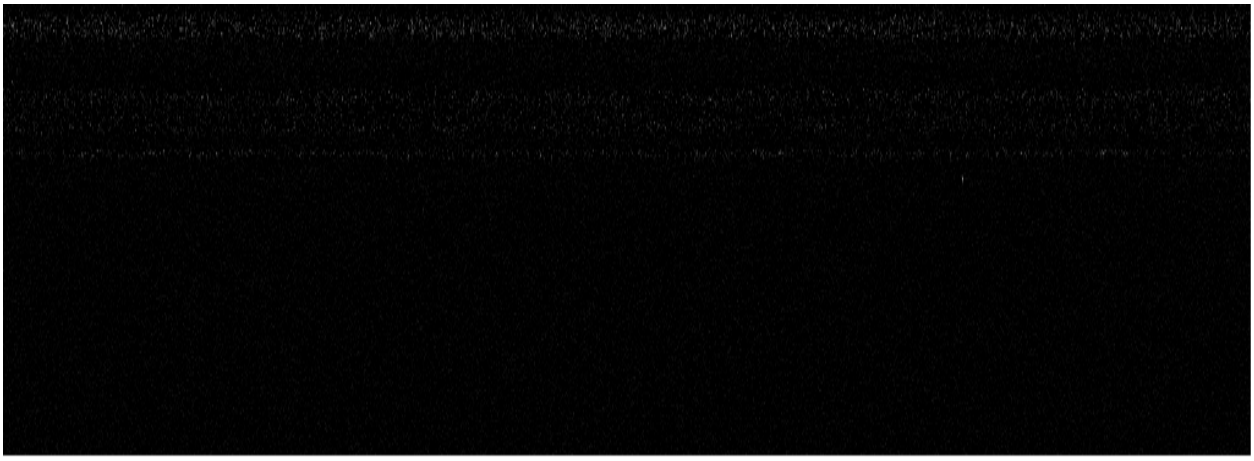

Figure C.4: Another example of an image misclassified by the Neural Network

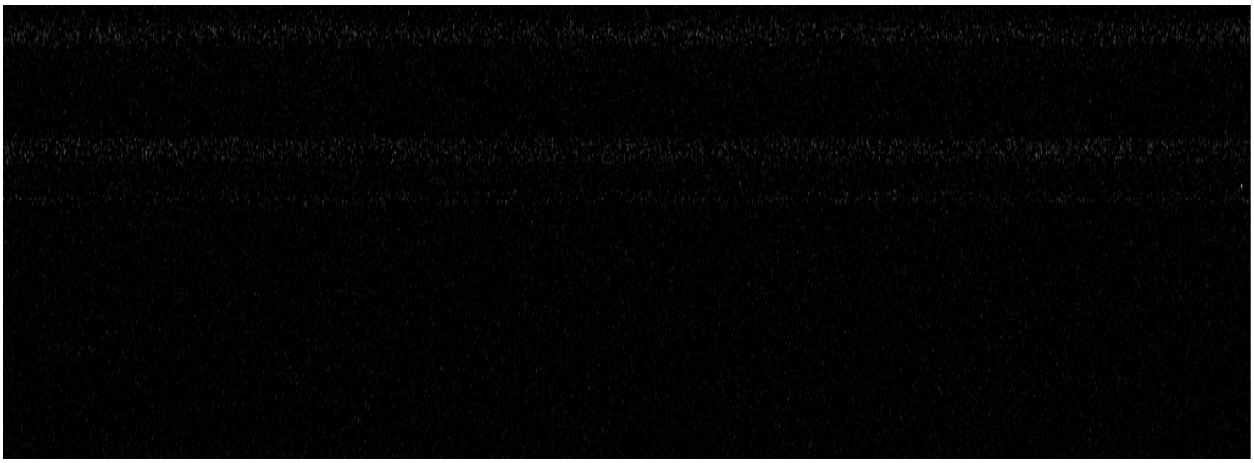

Figure C.5: Example of an image misclassified by all algorithms.

Figure C.6: Another example of an image misclassified by all algorithms.

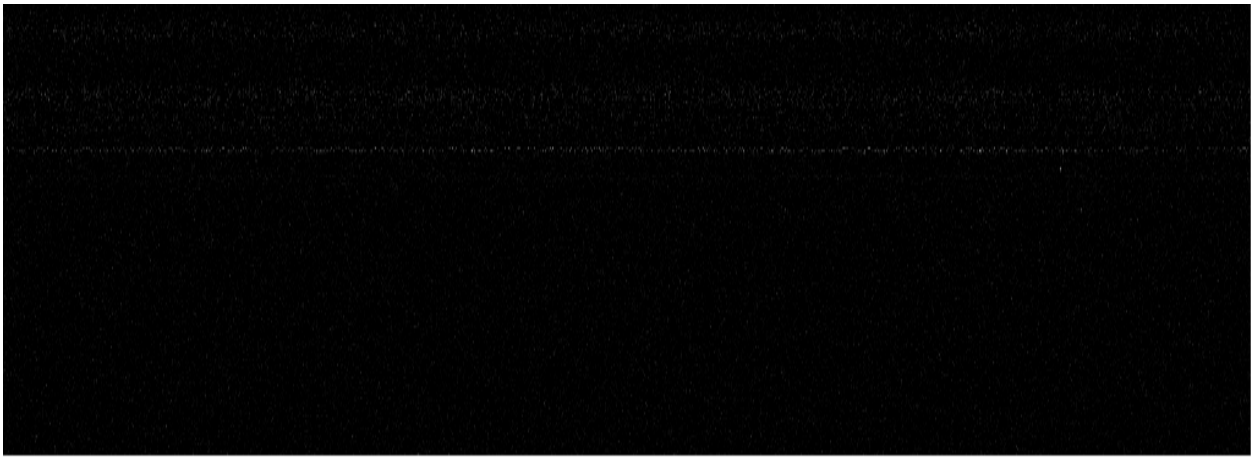

Figure C.7: Another example of an image misclassified by all algorithms.

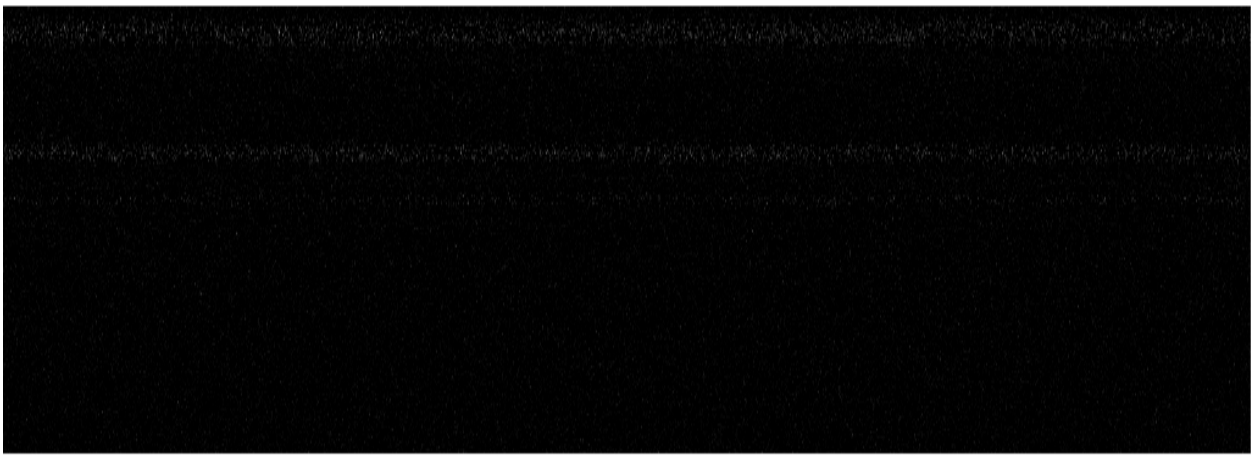

Figure C.8: Example of an image misclassified by AdaBoost and the Neural Network.

Figure C.9: Another example of an image misclassified by AdaBoost and the Neural Network.

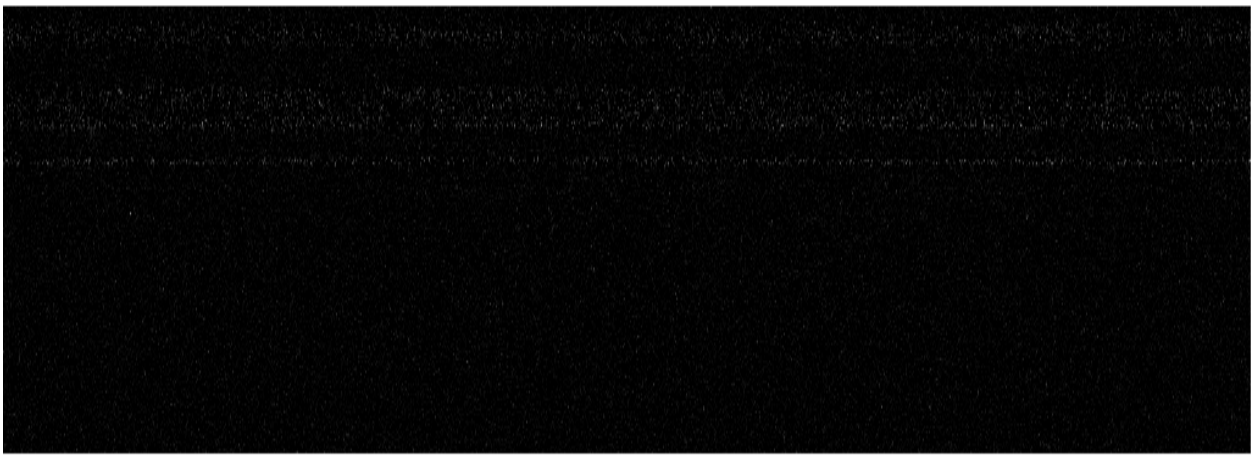

Figure C.10: Another example of an image misclassified by RUSBoost.