# An Ensemble-Based Approach for Classification of High-Resolution Satellite Imagery of the Amazon Basin

Anne He
anne.he@live.com

Jennifer He
jenhe2000@gmail.com

Richard Kim
richard.j.kim17@gmail.com

Daniel Like
dlike230@gmail.com

Arthur Yan
arthuryan1@gmail.com

Anthony Yang
yanga23@gmail.com

New Jersey Governor's School of Engineering and Technology

21 July 2017

***Abstract* — Land cover mapping of the Amazon Basin provides valuable insight for assessment of regional land use and the extent of human encroachment, which can help environmentalists detect and respond to deforestation. Current methods rely on coarse-resolution imagery, which limits effectiveness at identifying small-scale deforestation. This paper proposes an ensemble-based method utilizing weighted voting of heterogeneous classifiers for accurate and scalable labeling of high-resolution satellite imagery of the Amazon rainforest. The ensemble learner relies on five different classification models trained in parallel on a publicly available dataset of images with crowdsourced labels, and resulting predictions are aggregated as features into an additional meta-classifier. Performance of the ensemble is compared against that of individual classifiers on a hold-out test dataset.**

## I. Introduction

The Amazon rainforest is the largest and most extensive rainforest in the world, comprising over half of Earth's rainforest biome with a total land area of more than two million square miles. It is home to tremendous biodiversity, supporting an estimated one-quarter of all known terrestrial plant and animal species [1]. Moreover, the Amazon absorbs more than a half billion metric tons of carbon per year alone, playing a crucial role in the maintenance of the global carbon budget. Therefore, its preservation is an issue of great importance and urgency, as it is currently threatened by widespread deforestation, including heavy logging and agricultural clearance [3].

Deforestation has decimated large swaths of the Amazon, decreasing biodiversity and accelerating the rate of climate change. Currently, local governments and conservationists are ill-equipped to effectively address and combat an issue of this magnitude. In order to battle against deforestation, it is important to understand where it takes place. Active monitoring of the Amazon rainforest may provide valuable insight into where deforestation "hot spots" occur, and allow conservationists to identify indicators of deforestation, such as illegal mining operations or selective logging practices. However, this typically requires significant resources and a degree of regional coordination that has not yet been achieved. Ground-level land surveying is time-consuming and labor-intensive. It is currently not feasible as a long-term solution for monitoring the health and status of the Amazon ecosystem [6]. Thus, a possible effective solution could be an automated classification system capable of labeling indicators of deforestation solely from satellite imagery and remote sensing data.

Most current methods for tracking environmental change or classifying land cover and usage rely heavily on remote sensing data for airphoto interpretation [7]. However, the vast majority of classifiers developed in the past few decades are trained solely on coarse or medium-spatial resolution satellite imagery, primarily sourced from optical imaging satellite systems such as the Landsat Thematic Mapper (TM) and the SPOT High Resolution Visible (HRV). These satellite systems typically have panchromatic and multispectral resolutions ranging anywhere from tens to hundreds of square meters per pixel. Such resolutions have previously proven to be insufficient or inadequate in the classification and labeling of very detailed or localized objects in scenes. For example, data from the TM had greater spectral resolution than the HRV, and therefore was demonstrated to be more effective in differentiating species-level vegetation for detailed vegetation studies[8][9], but models utilizing either data type were unable

to achieve greater than 40 percent classification accuracy for thematic information extraction [13].

In general, *medium spatial resolution (MSR) satellite imagery*, defined as images containing pixels that have a ground sample distance (GSD) of 10-30 meters, are considered spatially and spectrally too coarse to distinguish the fine gradients between different clusters of vegetation units in an ecosystem [10]. With MSR images, a single pixel may encompass multiple distinct objects of interest. This results in a mixed field signature that is the result of an average of local objects, which may lead to difficulties in classification due to natural variations in the dataset. In contrast, high resolution images offer rich spatial, spectral, geophysical, and contextual scene information that cannot be matched by coarser resolution datasets.

There is an apparent need for high spatial resolution (HSR) data in many classification applications [11][12]. Similarly, the lack of HSR imagery of the Amazon Basin would render the detection of small-scale deforestation solely from remote sensing data exceedingly challenging.

Fortunately, HSR data has become increasingly available in recent years due to advances in remote sensing technology. Large volumes of airborne and spaceborne multispectral imagery can now be obtained at spatial resolutions at or better than one meter [12].

In 2017, Planet Labs, an Earth-imaging company, released a dataset of HSR satellite imagery of the Amazon to the public domain. This paper seeks to utilize this dataset to explore the implementation of an ensemble classifier based on weighted majority rule and stacked generalization for classification and labeling of satellite image scenes in order to detect signs of deforestation and map environmental trends. High resolution remote sensing data has been demonstrated as a powerful source of data for land cover and use monitoring, but it is currently limited by the lack of robust image segmentation methods and robust classifiers. The ensemble incorporates the predictive outputs of multiple heterogeneous classifiers, including convolutional neural networks, random forests, support vector machines, gradient boosting trees, and k-nearest neighbors, as features into a meta-classifier. This provides a robust and novel approach to classification of high-resolution satellite imagery as well as a significant contribution to monitoring environmental change and deforestation within the Amazon rainforest.

## II. Background

### A. Machine Learning

Machine learning is a process that allows computers to gain insight into data through identification of patterns and characteristics within the data, often in ways that humans may not notice. This paper will primarily use supervised machine learning to identify patterns and characteristics, specifically by using image data. Supervised learning involves the use of several classifiers, which are trained using pre-labeled images and use their training to identify similar characteristics in new images.

### B. Feature Descriptors

The simplest way to train image classifiers is to use raw, flattened image data. However, this method usually lacks accuracy, and therefore, most classifiers use feature descriptors. Feature descriptors are lists of values that represent various details about images, such as edges or the distribution of pixel gradients within the image.

*1) Canny Edge Detection:* The process of Canny Edge Detection begins by applying Gaussian blur to an image, which removes any noise that could interfere with edge detection [25]. The next step in the Canny Edge Detection algorithm is to find the derivative of the image. The derivative is the change in color between pixels, at each point in the image and in each direction. These two derivatives are represented respectively by $G_x$ and $G_y$. The formulas below display how these derivatives are used to calculate the magnitude and direction of an image's edge gradient at a given pixel. Once gradients are calculated, the Canny Edge Detection algorithm determines the placement of each gradient's magnitude between an upper and lower threshold. Automatically, all gradients with magnitudes above the upper threshold are classified as edges, while all gradients with magnitudes below the lower threshold are classified as non-edges. For gradients between the two thresholds, pixels are classified as edges based on whether gradients are continuous between adjacent pixels.

$$\text{Edge Gradient (G)} = \sqrt{G_x^2 + G_y^2} \tag{1}$$

$$\text{Angle}(\theta) = \arctan(\frac{G_y}{G_x}) \tag{2}$$

*2) Histogram of Oriented Gradients (HOG):* A histogram of oriented gradients is the distribution of the directions of gradients in an image. Gradients are the derivatives of the image matrix at each point, and often represent features like outlines or corners in an image. HOG has been successfully used for image classification in the past.

### C. Classifiers

*1) Support Vector Machines:* Support Vector Machine (SVM) is a machine learning classifier that classifies image vectors into categories based on training with labelized vectors of the same dimension. This involves finding the optimal hyperplane to divide the vectorized data. For example, if images in a training set were represented by a 200-dimensional vector, the SVM algorithm would construct an 199-dimensional hyperplane to split the training data, and would then use this hyperplane to make distinctions between images in testing

data. The SVM algorithm constructs this hyperplane by maximizing a quantity known as the *margin* of the training data. This quantity is equal to the minimum distance between the hyperplane and the training examples, calculated with this formula:

$$\text{distance} = \frac{| \beta_0 + \beta^T x |}{\| \beta \|} \tag{3}$$

The SVM algorithm maximizes this distance because a hyperplane that is too close to datapoints from the training set means that the classifier will be *noise-sensitive*, meaning that it will be too sensitive to small variations in training data. The notation used to describe a hyperplane is:

$$f(x) = \beta_0 + \beta^T x \tag{4}$$

where $\beta$ is known as the *weight vector* and $\beta_0$ is known as the *bias vector*. The optimal hyperplane to be used for an SVM is the scaled combination of these two vectors. $\mathbf{x}$ represents the training examples closest to the hyperplane, also known as *support vectors*. SVM has been shown to perform well in image classification tasks that are traditionally difficult with other classifiers [22].

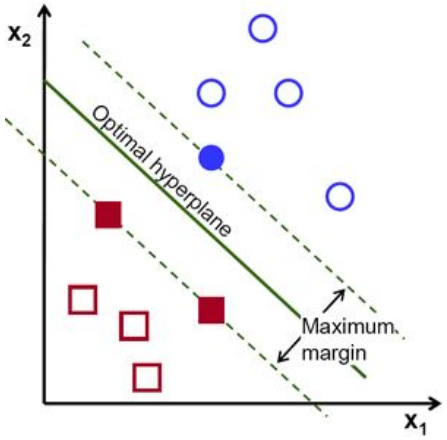


Fig. 1 An diagram of how an SVM classifies a set of linearly separable 2D-points.

*2) Convolutional Neural Networks:* Deep neural networks capture patterns in data at increasingly high levels of abstraction [19]. The models are organized into a series of layers, each of which learns a distinct representation of the dataset. Convolutional neural networks are based on layers that examine localized clusters of features, making them well-suited to image recognition tasks. Each CNN layer consists of a set of filters and a pooling layer that prevents overfitting by aggregating outputs from filters that are close to one another. The operations of a CNN layer can be written succinctly as:

$$\mathbf{O}^l = \text{pool}_P(\sigma(\mathbf{O}^{l-1} \star \mathbf{W}^l + \mathbf{b}^l)) \tag{5}$$

where $O^{l-1}$ is the input feature map to the lth layer, $\{W^l, B^l\}$ is the set of learnable parameters (weights and biases) of the layer, $\sigma(\cdot)$ is a logistic function that allows the neural network to learn nonlinearities of data, pool is the pooling operation, P is the size of the region of localized clusters that the pooling layer considers, and the star denotes a linear convolution operation.

*3) Random Forest Classifier:* Random forest classifier is a classification algorithm that takes the results of multiple decision trees from different subsets of a dataset to compute the final result, as shown in Figure 2. Decision trees, an integral part of the Random Forests Classifier, are predictive models that use binary logic to make decisions that categorize data. Data is evaluated through binary decisions, such as whether a value is greater than or less than a certain threshold, resulting in the tracing along the nodes of a "tree." If a terminal node is reached, then the node is assigned to the respective class [21]. The Random Forest classifier uses results from multiple different decision tree models, making it a meta-classifier that is a more robust model than just an individual decision tree model. Random Forest classifier improves accuracy and reduces overfitting by using random decision trees and averaging. Since it is competent at dealing with outliers in training data, this classifier has become increasingly popular for satellite and aerial image classification [22].

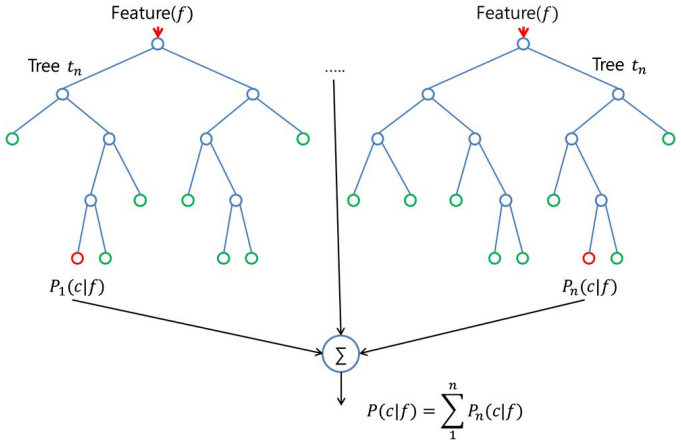


Fig. 2 Architecture of random forest ensemble composed of decision trees

*4) Gradient Boosting Trees:* Gradient Boosting Trees is an ensemble technique, meaning that uses weak learners such as decision trees to create an optimized model. This technique

iteratively runs through decision trees, each time adjusting new trees to the remainder of the previous. Through continuous self-adjustment, mean-squared error, a typical calculation of error in data classification, is minimized. The user can easily adjust how many iterations and trees a model needs based on their available computational resources and desired minimum error. Gradient Boosting Trees uses gradient descent, meaning that it approximates every step in a gradient, or slope, and moves in the direction of the minimum to reach the global minima. This helps the classifier reach an accurate result as quickly as possible. Gradient descent minimizes complicated loss functions as efficiently as possible and reduces bias in the dataset. A popular gradient boosting model that is implemented in this paper is Extreme Gradient Boosting, or XGBoost. This machine learning method is extremely popular in solving problems regarding data analytics due to its ease in handling large datasets quickly [18].

*5) K-Nearest Neighbors:* K-Nearest Neighbors (KNN) is a simple algorithm that stores all available training cases in order to construct the classification model. The training data is first plotted on an n-dimensional space where n is the number of data-attributes. Each point in n-dimensional space is labeled with its class value. The methodology then classifies new cases by finding a preassigned arbitrary number, k, of the most similar data points as nearest neighbors to a given target. The new point is plotted on the n-dimensional space and class labels of nearest k data points are noted. That class which occurs for the maximum number of times among the k nearest data points is taken as the class of the new data-point. The similarity between points can be defined by a measure of distance. [20]

### D. Ensemble Learner

An ensemble learner is a machine learning classifier that creates multiple hypotheses regarding the same question through different methods and then combines these hypotheses. Ensemble learners are superior in accuracy to individual learners, as they aggregate the results produced by multiple learners [4]. To produce the results used in this paper, an ensemble learning technique known as *weighted voting* or *model averaging* was employed. For this process, the decisions of each classifier, known as a base learner, are given a weight representative of the accuracy of the classifier determined during the training process. For the purpose of this paper, each classifier's weight was set equal to its $F_\beta$ score.

### E. Python Programming

*Python* is an object-oriented programming language with a built-in emphasis on code simplicity, providing it an advantage over other languages like Java and C++. As a general-purpose programming language, Python is extremely versatile and has many applications, especially when used with the correct tools and libraries. To achieve the computer vision and machine learning objectives of this paper, Python was used in conjunction with OpenCV, scikit-image, and scikit-learn. OpenCV is an open-source computer vision library that provides methods for computer vision-related tasks such as image classification and feature detection. Scikit-image, similar in purpose to OpenCV, is also used for image feature extraction.

Scikit-learn is another open-source library for Python. It is commonly used for any kind of application related to machine learning. It provides multiple classification algorithms including support-vector machines, random forests, gradient boosting, k-nearest neighbors [24].

## III. Procedure: Creating And Training Classifiers

### A. Satellite Imagery Data

The dataset used for training and evaluation of classification models was obtained from "Planet: Understanding the Amazon from Space," a publicly available competition on Kaggle, a platform that hosts data science competitions and publishes datasets. The dataset used was composed of HSR satellite images (orthorectified pixel size of three meters) collected by Planet Labs' Flock 2 satellites in sun-synchronous orbit (SSO) and International Space Station (ISS) orbit between January 1 and February 1, 2017. Satellite images were collected over multiple South American countries, including Brazil, Peru, Uruguay, Colombia, Venezuela, Guyana, Bolivia, and Ecuador.

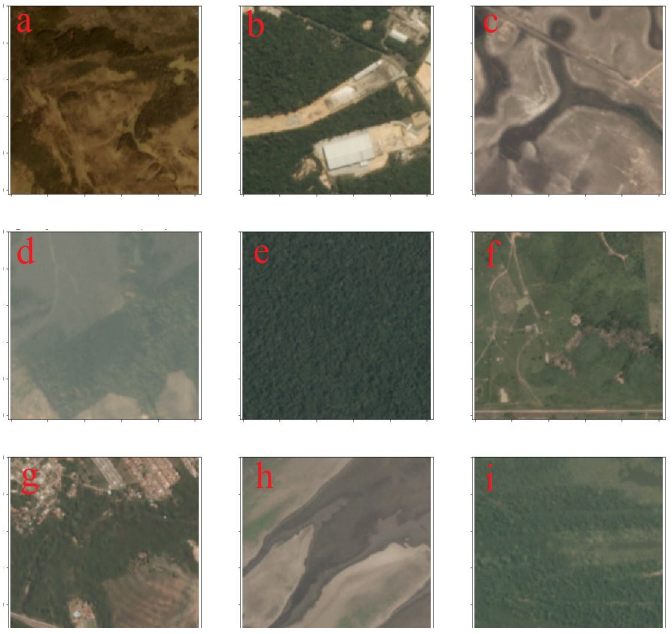


Fig. 3 Examples of labeled satellite image chips from train-jpg depicting various states of land usage and cover: (a) bare ground; (b) habitation, road, and water; (c) bare ground and road; (d) agriculture, road, water, and

cultivation; (e) primary rainforest; (f) and (g) agriculture, habitation, and road; (h) water; and (i) blooming.

The complete dataset consists of over 150,000 distinct 256 x 256 multispectral satellite chips derived from over 1,600 scenes, covering a total land area of over thirty million hectares. Each satellite chip is stored in the GeoTiff format and contains four bands of data: red, green, blue, and near infrared. Channels are encoded as 16-bit digital numbers.

For the purposes of the competition, the authors of the data have stripped all GeoTiff information from the dataset, including data regarding chip footprint and ground control points, which are points with known location used for geo-referencing. This forces classifiers to extract relevant features from raw image data rather than to rely on geotag information.

Class labels were chosen to represent a reasonable subset of phenomena of interest in the Amazon basin and either describe atmospheric conditions or describe land cover and land use phenomena. Each chip has at least one atmospheric condition label attached to it and may or may not have any land cover or use labels. Examples of atmospheric condition labels include cloudy, partly cloudy, haze, and clear. Cloud cover labels play a significant role in classification, as a completely cloudy or hazy image presents a major challenge for passive satellite imaging, often resulting in an obscured image with no observable ground-level characteristics.

Land cover and use labels are broken down based on how frequently they occur. Common labels, such as primary rainforest and water, are relatively straightforward to predict due to their size and characteristics, as they tend to dominate images they are associated with. Rarer labels such as selective logging and blooming are typically associated with more interesting or alarming phenomena but are harder to classify due to the relative lack of training examples. In general, however, both types of labels are important for interpretation of results and deriving insight into the mechanics behind environmental change and deforestation.

Data labeling was done via crowd labeling through the CrowdFlower platform in conjunction with analyst teams from Planet. As a result, chip labels are inherently noisy due to the ambiguity of features and the labeling process. Certain scenes may omit class labels or have incorrect ones. However, the authors believe that their data has a high signal-to-noise ratio and is well-suited to classification models.

In addition to the TIF images provided, the authors have also included JPG chips for reference and practice. The TIFs contain multispectral information that is not present in the JPGs, so it is expected that utilizing them would result in better classification accuracy, but they were mislabeled at the time of writing this paper. Due to memory constraints, the classifiers and ensemble were trained and tested on the JPG chips.

The dataset was distributed in two parts: train-jpg and test-jpg. Labeled training images were provided in train-jpg and unlabeled test images were provided in test-jpg. Labels were stored as CSV file.

### B. Evaluation Metrics

The main evaluation metrics used were precision, recall, $F_\beta$ score, and accuracy. *Precision* is the measure of the fraction of positive predictions that are correct. This was determined by calculating the sum of the true positives, or the positives the classifier predicted correctly, then dividing the sum by the sum true positives and false positives, or all of the positives predicted.

*Recall* is the measure of the fraction of correct positive predictions out of all actual positive entries. This was determined by calculating the sum of true positives, then dividing the sum by the sum of true positives and false negatives, or all entries that were actually positive.

*$F_\beta$ score* is a measure of accuracy based on both precision and recall. Beta is a constant that changes the coefficient of the formula. Increasing the value of $\beta$ increases the weight of recall in the $F_\beta$ score. As seen in Fig. 2, $F_\beta$ score is calculated by multiplying the sum of one and $\beta$ squared by precision and recall, then dividing it by the sum of recall and the product of $\beta$ squared times precision.

The $F_\beta$ scores within this paper were $F_2$ scores, meaning that they utilized a $\beta$ value of two. In the case of $F_2$ scores, recall was prioritized. It was more important to predict a greater percentage of actual positives than to predict a smaller percentage of false positives.

*Accuracy* is a more general measure of performance. This was determined by dividing the sum of true positives and true negatives by the total number of outputs. Accuracy and precision values range from zero to one but represent a percentage of correctly classified images.

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}} \qquad (5)$$

### C. Feature Extraction

For feature extraction, OpenCV was originally used, but as was found through experimentation, Scikit-Image's feature extraction algorithms, such as HOG, produced vectors that were better suited for use in each classifier. To use Scikit-Image to generate HOG descriptors, each image needed to be converted to a grayscale image, and then rescaled in intensity. To generate flattened images, Scikit-Image's flattening function was used on each of the training and testing images. To generate Canny Edge Detection descriptors, OpenCV was used to generate an image with each image's edges, which was then flattened.

### D. Training Models

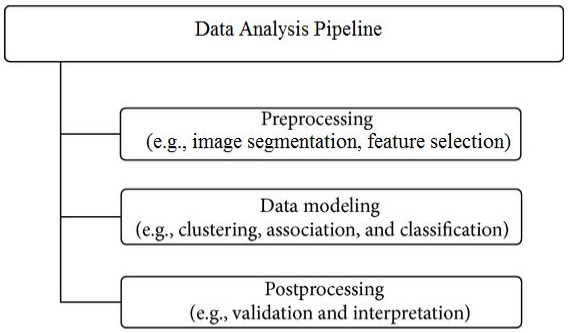


Fig. 4 Data analysis pipeline.

To construct an ensemble learner with the optimized classification abilities, a variety of individual classifiers were investigated. Each classifier followed a general data analysis pipeline as seen in Figure 4. The first step was preprocessing, in which different feature detectors and image reshaping took place. Once completed, the model was trained and tested to create a complete classifier. The results were recorded through accuracy measurements of the classification task.

*1) SVM*: To build SVM classifiers, training data was first vectorized through feature detection or other means, as the SVM algorithm can only deal with data in the form of n-dimensional vectors.

The first SVM classifier was trained with raw image data, without any feature extraction. Because images are traditionally represented as matrices, they needed to be flattened to vectors. First, colored satellite imagery was converted into grayscale. Colored imagery has three channels of values: red, green and blue. Meanwhile, grayscale imagery only has one channel, brightness. When the images were converted to grayscale, the dimensions were lowered enough to allow conversion into vectors.

After classification using raw image data, SVM was switched to classifying images using various feature descriptors. Similar to the vectorization of imagery, the algorithms producing feature descriptors required grayscale imagery. The first feature descriptor used was Histogram of Gradients (HOG). For each image, a set of HOG descriptors- represented as a vector- was produced by the scikit-image library and subsequently used to train an SVM model. This was also attempted with Oriented FAST and Rotated BRIEF (ORB), another feature descriptor, which was also generated using the scikit-image library. However, ORB relies upon significant differences in intensity between adjacent pixels, which the satellite imagery lacked. Therefore, ORB could not be used for this experiment without extensive image preprocessing.

An additional feature descriptor used to train SVM models in this experiment was edge detection. To train with this descriptor, OpenCV's Canny Edge Detection algorithm was used to generate an image of each picture's edges. These images of edges were then flattened into vectors in the same manner as that of raw images.

For each of the above methods of training SVM models, multiple binary SVM classifiers were built to form a multi-label SVM. Each of these classifiers was responsible for determining the existence of one characteristic in given satellite imagery. For example, for each type of image-describing vector, one SVM classifier was built to classify whether or not images displayed land cultivation, and a separate SVM classifier was built to classify whether or not images displayed conventional mining.

*2) CNN:* Convolutional neural networks were implemented using Keras 2.0.5 with Theano 0.9.0 as a backend for matrix- and tensor-based computations. Training time was accelerated with a Tesla K80 GPU on the XStream Cray CS-Storm Cluster at Stanford Research Computing Center.

Early versions of the CNN were trained on the train-jpg dataset with the first 35,000 values used as training data and the remaining 5,479 images as the validation set.

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
activation='relu', input_shape=(32, 32, 3)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(17, activation='sigmoid'))
```

Fig 5. Code snippet for baseline convolutional neural network, written in Python with Keras 2.0.5.

Images were resized to 32 x 32 in order to reduce training time for prototyping. Image sizes of 64x64, 128x128, and 224x224 were also tested on this initial model in order to determine the impact on classifier performance. A MaxPooling layer was added to reduce overfitting; this layer combines the outputs of several clustered neurons in prior layers into a single neuron in the following layer, which reduces the dimensionality of learned weights. The CNN also utilized Dropout layers, which randomly set the weights of connections between layers to zero to reduce the number of parameters learned and prevent overfitting. Dropout rates were initialized at 0.5, meaning that each connection had a one in two chance of being randomly dropped. Dropout rates of 0.25 and 0.75 for both dropout layers were also tested. This initial model was primarily used to fine-tune hyperparameters, as the training time was only ~26 seconds per complete pass over the entire dataset. Different numbers of filters in the second convolutional layer and different kernel sizes were also tested. Early models containing only 2 convolutional layers were run for a fixed 15 epochs without early stopping. An epoch is defined as one complete pass over the dataset; 15

was chosen randomly to ensure sufficient and reasonable training time.

```
model.add(Conv2D(64, kernel_size=(3, 3),
padding='same', activation='relu'))
    model.add(Conv2D(64, (3, 3),
activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))
```

Fig 6. Code snippet for convolutional blocks used to construct final deep CNN model.

In later models, six additional convolutional layers were added, along with one max pooling layer and one dropout layer per pair of convolutional layers. These were added in identical blocks, as shown in Figure 6. Each set of layers had identical hyperparameters except for the number of filters, which doubled in each consecutive set of layers. The model was later extended to the test set and hold-out validation was replaced with k-fold cross-validation, an approach that allows the model to train on all the data. In k-fold cross-validation, the complete train-jpg dataset was segmented into k equally sized partitions and k different models were trained, each using a different partition as the sole hold-out validation set. Then, the models are averaged together to produce a more robust CNN.

F2 threshold optimization by brute-force search was also implemented. The performance of all classifiers was measured primarily through F2 score and decisions are made based on F2 threshold values. However, the best threshold values frequently vary between distinct classes, which is especially pronounced in a multi-class classification problem with uneven class distribution, so it is beneficial to classification performance to compute each one independently.

*3) Random Forests:* The Random Forests Classifier is most commonly implemented using Python and R, but in this case it was only implemented in Python to keep the programming language consistent with the other models.

A Random Forests Classifier was created from the sklearn.ensemble library and includes parameters. Next, a function was created for feature extraction, which converted the image to the correct format for the Random Forests function.

Similar to SVM and XGBoost, training data had to go through feature extraction. Various methods such as image flattening, edge detection, and HOG were implemented separately with the Random Forests classifier, and the performance of each model was compared.

For each of the above methods of training Random Forests models, multiple Random Forests classifiers were built. As with SVM, each of these classifiers was responsible for determining the existence of one characteristic in given satellite imagery. For example, one Random Forest classifier was built to classify whether or not images displayed land cultivation, and a separate Random Forest classifier was built to classify whether or not images displayed conventional mining.

Next, a function was created to assign a boolean value to each of the images for each of the 17 classifiers. A function was created to train the data. The dataset was split into a training set and a testing set. ⅞ of the dataset of 40,479 images was used for training and the remaining ⅛ was used for testing. Lastly, the code compared the predictions to the actual results and values for accuracy and $F_\beta$ score were calculated and displayed.

Subsequently, other image processing methods including edge detection and histogram of oriented gradients were used instead of image flattening in an effort to maximize accuracy and see what works well with Random Forest classifier. For the model that used HOG as an image processing method, a set of HOG descriptors, represented as a vector, was produced by the scikit-image library and subsequently used to train a Random Forests Classifier model. To train with the edge detection descriptor, the OpenCV library's Canny Edge Detection algorithm was used to generate an image of each picture's edges. These images of edges were then flattened into vectors in the same manner as that of raw images. After that, the same steps as the those for the original Random Forest classifier were repeated for these models. Each model was tested on a small subset of the data to emulate legitimate results for comparison. Once the optimal function was chosen, that model was further tuned to improve results.

In addition, the parameters of the classifier were altered independently to see what yields better results. Parameters that define the Random Forests Classifier include n_estimators, max_features, criterion, max_depth, min_samples_split, and min_samples_leaf. These parameters define the characteristics of the decision trees and the forest. For example, the parameter n_estimators is an integer that represents the number of decision trees in the forest. While the default value for this is 10, it was found through trial and error that a value around 20 was more suitable for our algorithm and produced better results and higher scores. This type of parameter tuning was also applied to other parameters of the classifier. The final Random Forests model with tuned parameters was run on the full dataset and results analyzed and output file added to ensemble learner.

*4) XGBoost:* Similar to SVM, training data had to be vectorized before building an XGBoost model using OpenCV's image reading function. This data was converted and stored in NumPy arrays so that they could be processed later on and input into the classifier model.

The initial model of the XGBoost classifier utilized only the raw image data. The images were resized from a size of 256 by 256 pixels to 32 by 32 pixels using OpenCV. The flattened pixels were flattened into a 3072 cell vector and directly input as features in the classifier.

Because the first instance of the XGBoost model was mainly meant to test its functionality, all parameters were set to default. However, to accommodate both the multi-class and multi-label qualities of the task, more modifications had to be made. XGBoost did not inherently support multi-label classification, so one vs. rest, also known as one vs. all, was investigated. One vs. rest is a scikit-learn function that splits a multi-label classifier into separate binary classifiers. To build XGBoost into one vs. rest, XGBoost had to be built using the scikit-learn wrapper in the form of an XGBoost Classifier. The reason for this is that scikit-learn's XGBoost Classifier takes in image data and label data in two separate arrays, while XGBoost's training method takes in a single matrix of both data and labels, limiting the data to a NumPy 2D array and labels to a 1D array.

After creating the XGBoost Classifier, it was cast as a OneVsRestClassifier to allow the model to handle multi-label processes. The model was then fit to the data, which allowed the model to make predictions and output measures of prediction performance. Issues were encountered when combining existing performance measuring methods with the multi-class multi-label model. Most methods that evaluate classifier performance, specifically those within the scikit-learn library, are not designed for multi-label or multi-class classification, and even fewer both. A new method for evaluation was written specifically for this model.

To optimize performance evaluation, performance was analyzed for each class and averaged using a micro average. A *micro average* is an average that aggregates all the individual true and false positives and negatives, which prevents overrepresentation of certain smaller classes in the dataset. Preventing this overrepresentation showed that there were some labels, such as slash-and-burn, that were very rare and difficult to account for.

A major challenge for this classifier was the time it required to train. The average run time was about seventy minutes, during which no modifications could be made and saved to the model. Because image data is very complex and XGBoost relies on an intricate structure of decision trees, it needs an extensive amount of time to completely load the data and train the model. The final prediction for the first XGBoost model returned an average precision score of 0.793 and accuracy score of 0.9262.

Modifying the parameters of XGBoost using cross-validation was also challenging. The usage of stratified k-fold with multi-label and multi-class classifiers is currently an unresolved problem in the field of machine learning [16]. Therefore, the classifier had to be split into separate binary classifiers.

After implementing a model based on flattened image data, other feature extractors were experimented with. HOG and Canny Edge Detection were used to generate feature descriptors, but these two algorithms did not produce any notable improvements and slowed down the model,so they were ultimately abandoned.

*5) KNN:* In order to implement the KNN Classifier, the training data was vectorized using the OpenCV library. Because the raw images were found to be too large for the model to handle in a time efficient manner, the images were resized to a 64 by 64 pixel file using an OpenCV function. The training image vectors were then flattened into 2 dimensional NumPy arrays to allow for simpler processing for the classifier model.

The 40,479 training images were split into a training set and a validation set, with seven-eighths of the images sorted into the former.

The current KNN model was created as a baseline classifier using default parameters, with k = 3. The default measure of distance, or the distance metric, between two data points was measured using the Euclidean distance formula. This formula measures the "ordinary" straight-line distance between two points in Euclidean space. Thus, the classifier used this formula to determine the distance in the feature space from a test data point to each training value.

$$D(a,b) = \sqrt{\sum_{i=1}^{n} (b_i - a_i)^2} \tag{6}$$

This task entails multi-label and multi-class classification capabilities, meaning the dataset contains pictures that may be assigned multiple labels and more than two different satellite chip labels. The KNN was altered to support these features. Using the one vs. rest classification function from the Scikit Learn library, the classifier was able to deal with multiple different classes and assign satellite images more than one tag if deemed necessary.

The trained model was validated by the test set of satellite chips. In order to evaluate the classifier performance, a series of evaluation metrics were calculated, including the aforementioned precision, recall, and $F_\beta$ score.

*6) Ensemble Learner:* Weighted voting ensemble learning was carried out by first marking each classification made by individual learners with either a 0 or a 1. A 0 was set to mean that the classifier had determined an image to not have a given characteristic and a 1 was set to mean that the classifier had determined an image to have a given characteristic. Each of these numerical classifications was then multiplied by the $F_\beta$ score of the classifier
that had made the classification for the specific characteristic being tested. These weighted decision values were then summed for each decision and divided by the sum of each classifier's $F_\beta$ score.

$$\text{Decision} = \frac{\sum_{i=1}^{n} f_i d_i}{\sum_{i=1}^{n} f_i} \quad (7)$$

The above formula can be used to represent the weighted decision made for a specific feature for a specific decision, where $f_i$ represents the $F_\beta$ score of a classifier for the characteristic being predicted. For example, if one classifier with an $F_\beta$ score of 0.8 for slash-and-burn had classified a specific image as 0 for that characteristic and another classifier with an $F_\beta$ score of 0.9 for the same characteristic had classified the same image as 1, then the total weighted decision for slash-and-burn for that image would be 0.9 (0.8*0 + 0.9*1), which would then be divided by the $F_\beta$ score total of 1.7, resulting in a value of approximately 0.53. This would then be rounded, resulting in a final decision of 1 for the characteristic slash-and-burn for that specific image.

## IV. Results and Discussion

### A. SVM

When each SVM was tested on the testing subset of the training set, varying accuracy values were measured. Table 1 summarizes these accuracy values for SVMs trained with each feature extractor.

Table 1. SVM Results with Different Feature Extractors

| **SVM Results** | | |
|---|---|---|
| **Feature Extractor** | **$F_\beta$ score** | **Accuracy** |
| HOG | 0.61846 | 0.76286 |
| None (flattened image) | 0.62899 | 0.89863 |
| Canny Edge Detection | 0.63030 | 0.90655 |

As can be seen in Table 3, Canny Edge Detection produced both the highest accuracy (91%) and $F_\beta$ score (0.630) when used to produce training data for SVM. A flattened image came in second for both metrics as well (accuracy of 90% and $F_\beta$ score of 0.629). The use of HOG descriptors led to the lowest accuracy and $F_\beta$ score (76% and 62% respectively), but the method had a much lower execution time.

### B. CNN

The baseline model with two convolutional layers, a (3, 3) kernel size, and 64 filters in the second layer achieved 93.16% accuracy on a holdout test set with 0.1756 validation loss. F2 score was not recorded. Input images were resized to 32x32 to reduce model complexity and train time. Reducing kernel size to (1, 1) in subsequent iterations of the model helped convolution filters better capture representations of localized statistical features. Increasing the batch size to 256 from 128 and the number of filters in the second convolutional layer to 128 from 64 increased the number of parameters learned helped the model achieve a better fit to the data and improved performance significantly at the cost of greater memory and time constraints. Training time increased to 26 seconds from 21 after implementing these changes. This model achieved 94.06% accuracy on a holdout test set with 0.1514 loss and an F2 score of 0.8591. After adding six more convolutional layers and implementing 3-fold cross-validation as well as F2 threshold optimization via brute-force, accuracy improved to 95.00% with 0.1256 loss and F2 score of 0.8828. This model was also run on the test-jpg dataset to make predictions and received a 0.88316 F2 score on the Kaggle LeaderBoard. Implementing an adaptive learning rate (changes every 30 epochs) and a much longer training process, performance improved to roughly 95.81% accuracy with 0.1112 loss and 0.9032 F2 score. On the Kaggle LeaderBoard and test-jpg dataset, this model received an F2 score of 0.90462.

### C. Random Forest

Each model with different feature extractors was tested on the subset of the data to emulate legitimate results for comparison.

Table 2. Random Forest with Different Feature Extractor

| **Random Forest Results** | | |
|---|---|---|
| **Feature Extractor** | **$F_\beta$ score** | **Accuracy** |
| HOG | 0.6235 | 0.9049 |
| None (flattened image) | 0.6690 | 0.9148 |
| Canny Edge Detection | 0.6182 | 0.9096 |

By comparing the outputs, it was clear that the model that flattened the images produced the highest accuracy and $F_\beta$ score at the same time. This model run on the full dataset yielded 93% accuracy with a 0.71 $F_\beta$ score. The flattened images produced more favorable results because the images were processed the least, and useful information for the classifier was not removed. The overall favorable result for Random Forests compared to other classifiers is due to its structure as an ensemble learner which makes it more advanced than other algorithms.

### *D. XGBoost*

The first XGBoost model was successfully built and run on the entire training set of over 40,000 images. It trained on two-thirds of the set and tested one-third. It successfully predicted an accuracy of 0.9262 for test images while using multi-label input and multi-class output. The average precision score was 0.793. The second XGBoost model was split based on seven-eighths of the set as training and one-eighth as testing. This improved the accuracy by 0.07 percent for 92.69 percent. It returned a precision score of 0.8703, recall score of 0.6639, and F2 score of 0.6970. Even when images were resized to a smaller resolution of 32 by 32 pixels to optimize speed, the model still proved very accurate with simply raw image data.

Unlike the first two models, the third and fourth models did not use raw image data resized to 32 by 32 pixels. Because these models extracted features to be input into the model, the input array was smaller and images could be resized to 64 by 64 pixels before feature extraction.

TABLE 3. XGBOOST RESULTS WITH DIFFERENT FEATURE EXTRACTORS

| **XGBoost Results** | | |
|---|---|---|
| **Feature Extractor** | **$F_\beta$ score** | **Accuracy** |
| HOG | 0.6220 | 0.9102 |
| None (flattened image) | 0.6958 | 0.9262 |
| Canny Edge Detection | 0.6192 | 0.9070 |

As seen in Table 1, raw image data still proved to produce the highest accuracy and $F_\beta$ score, despite the input images being of a smaller size. The feature extractors likely processed out some necessary information as noise, while raw image data allowed the model to determine the most important features itself.

### *E. KNN*

The baseline model of the KNN was successfully trained and tested using the training image set using Canny Edge Detection as the feature detector. The model trained on the first seven-eighths of the data set, with the validation set being the remaining images. The model ran an accuracy value of 78% with an $F_\beta$ value of 0.68.

The major limiting factor in developing the KNN classifier was the large amounts of time required to train and test the datasets. The first iteration on the complete training data file took more than 50 hours to complete. In order to address this issue, KNN has various parameters that can be altered in order to improve both speed and efficiency. Implementing a ball tree algorithm may be a solution. The algorithm takes the unclassified data point and classifies it into one of two different ball shaped clusters in the multidimensional space based on the distance from the centroid of the ball. Then each ball is subdivided again into sub-clusters repetitively again in order to determine a small cluster of points that are closest to the data point. Because this eliminates having to check the distance of every single point in the multidimensional space, it speeds up the classifier by not requiring the classifier to check extraneous points.

Improving the accuracy measures was, and still is, an ongoing challenge left to be dealt with. K-fold cross validation may be implemented to solve this issue. By creating k subsamples that each may serve as a validation dataset for testing a model, this method combines the results of k different models because they were trained using different sets of data. This can deter overfitting while increasing efficiency.

### *F. Ensemble Learner*

The ensemble learner was built successfully, consolidating the results of the best model for each classifier. The ensemble learner constructed produced results with 97% accuracy and an $F_\beta$ score of 0.96. This represents a significant improvement, as most of the classifiers had accuracies near 90 percent and $F_\beta$ scores of around 0.7, with the exception of the CNN, which had an accuracy of 93 percent and an $F_\beta$ score of 0.9.

## IV. CONCLUSION

The combination of computer vision and machine learning has the potential to change the way the world tracks environmental changes. It can be seen that the key to improving machine learning models and current classification techniques is to create ensembles combining many different models. Machine learning can quickly process large amounts of data, even visual data like satellite imagery. The ensemble presented within this paper is a novel approach to extrapolating information from high-resolution satellite

imagery. The authors were restricted by the limited processing power of personal computers, but with greater computing power, the ensemble model could train and predict tens of thousands of images within only a few hours. If refinement is continued, the accuracy of the model could be implemented in real-world applications and used to discover the signs of rainforest loss so that it can be stopped or reduced. Not only could it be implemented with imagery of the Amazon rainforest, but also other rainforests or even other volatile biomes.

## Acknowledgments

The authors of this paper would like to gratefully acknowledge the assistance and guidance of project mentor Anthony Yang and residential teaching assistant Shantanu Laghate. The authors would also like to thank Dean Ilene Rosen, the Director of the NJ Governor's School of Engineering and Technology (GSET), Dean Jean Patrick Antoine, Associate Director of GSET, and the sponsors of GSET: Rutgers University, Rutgers School of Engineering, The State of New Jersey, Lockheed Martin, Silverline Windows, and GSET Alumni.

This work used the XStream computational resource, supported by the National Science Foundation Major Research Instrumentation program (ACI-1429830).

## References

[1] R. Dirzo, P. H. Raven. (2003) *Global state of biodiversity and loss*. Annu. Rev. Environ. Resour. [Online]. Available: http://www.annualreviews.org/doi/pdf/10.1146/annurev.energy.28.050302.105532

[2] G.J. Hay, D.J. Marceau, P. Dube, and A. Bouchard. "A multiscale framework for landscape analysis: Object-specific analysis and upscaling", *Landscape Ecology*, vol. 16, no. 6, pp 471–490, Aug. 2001.

[3] J. Knorn, A. Rabe, V. C. Radeloff, T. Kuemmerle, J. Kozak, and P. Hostert. "Land cover mapping of large areas using chain classification of neighboring Landsat satellite images," *Remote Sensing of Environment,* vol. 113, no. 5, pp. 957-964, 2009.

[4] Z. Zhou. "Ensemble Learning," *Encyclopedia of Biometrics*, pp. 411-416.

[5] M. A. Hearst, "Support vector machines," *IEEE Intelligent Systems, July/Aug. 1998, pp. 18-28. [Online]. Available:* http://www.cs.cmu.edu/afs/cs.cmu.edu/usr/guestrin/www/Class/10701/readings/hearst98.pdf

[6] T. N. Kehl, V. Todt, M. R. Veronez, and S. Cazella, "Amazon Rainforest Deforestation Daily Detection Tool Using Artificial Neural Networks and Satellite Images," *Sustainability*, vol. 4, no. 12, pp. 2566–2573, Apr. 2012.

[7] H. Sandmann and K. Lertzmann. (2003) *Combining High-Resolution Aerial Photography with Gradient-Directed Transects to Guide Field Sampling and Forest Mapping in Mountainous Terrain*. Society of American Foresters. [Online]. Available: https://www.researchgate.net/publication/233601532_Combining_High-Resolution_Aerial_Photography_with_Gradient-Directed_Transects_to_Guide_Field_Sampling_and_Forest_Mapping_in_Mountainous_Terrain

[8] D.J. Marceau, P.J. Howarth, J.M.M. Dubois, and D.J. Gratton. "Evaluation of the gray-level cooccurrence matrix method for land-cover classification using SPOT imagery", *IEEE Transactions on Geoscience and Remote Sensing*, vol. 28, no. 4, pp.513–519, 1990.

[9] P. Gong, and P.J. Howarth. "Land-use classification of SPOT HRV data using a cover-frequency method", *International Journal of Remote Sensing*, vol. 13, no. 8, pp. 1459–1471, 1992.

[10] P.F. Hsieh, L.C. Lee, and N.Y. Chen. "Effect of spatial resolution on classification errors of pure and mixed pixels in remote sensing", *IEEE Transactions on Geoscience and Remote Sensing*, vol. 39, no. 12, pp. 2657–2663, 2001.

[11] H. Nagendra, and D. Rocchini. "High resolution satellite imagery for tropical biodiversity studies: the devil is in the detail." *Biodiversity and Conservation*, vol. 17, no. 14, pp. 3431-3442, 2008.

[12] T. Whiteside, and R. Bartolo. "Mapping Aquatic Vegetation in a Tropical Wetland Using High Spatial Resolution Multispectral Satellite Imagery." *Remote Sensing*, vol. 7, no. 9, pp. 11664-11694, 2015.

[13] A.M.B. May, J. E. Pinder, and G. C. Kroh. "A comparison of Landsat Thematic Mapper and SPOT multi-spectral imagery for the classification of shrub and meadow vegetation in northern California, U.S.A." *International Journal of Remote Sensing*, vol. 18, no. 18, pp. 3719-3728, 1997.

[14] R.L. Czaplewski., and P.L. Patterson, 2003. Classification accuracy for stratification with remotely sensed data, Forest Science, 49(3):402–408.

[15] W. Wei, X. Chen, and A. Ma. "Object-oriented information extraction and application in high-resolution remote sensing image." *Proceedings. 2005 IEEE International Geoscience and Remote Sensing Symposium, 2005. IGARSS '05* (n.d.). doi:10.1109/igarss.2005.1525737.

[16] W.F. Laurance, C.A. Peres. Emerging Threats to Tropical Forests. Chicago: Univ Chicago Press; 2006.

[17] K. Sechidis, G. Tsoumakas, and I. Vlahavas. "On the Stratification of Multi-label Data." *Machine Learning and Knowledge Discovery in Databases*, 2011, 145-158. doi:10.1007/978-3-642-23808-6_10.

[18] T. Chen, and C. Guestrin. "XGBoost." *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*, 2016. doi:10.1145/2939672.2939785.

[19] A. Romero, C. Gatta, and G. Camps-Valls. "Unsupervised Deep Feature Extraction for Remote Sensing Image Classification." *IEEE Transactions on Geoscience and Remote Sensing* 54, no. 3, pp. 1349-1362, 2016.

[20] G. Chirici, M. Mura, D. McInerney, et. al. "A meta-analysis and review of the literature on the k-Nearest Neighbors technique for forestry applications that use remotely sensed data." *Remote Sensing of Environment* 176 (2016), pp. 282-294..

[21] A. Bosch, A. Zisserman, and X. Munoz, "Image Classification using Random Forests and Ferns," 2007 IEEE 11th International Conference on Computer Vision, 2007.

[22] N. Horning, "Random Forests : An algorithm for image classification and generation of continuous fields data sets," *GIS-IDEAS*, 2010.

[23] O. Chapelle, P. Haffner, and V. Vapnik, "Support vector machines for histogram-based image classification," *IEEE Transactions on Neural Networks*, vol. 10, no. 5, pp. 1055–1064, 1999.

[24] L. Buitinck, G. Louppe, M. Blondel, et al. "API design for machine learning software: experiences from the scikit-learn project." *arXiv preprint arXiv.*(2013)

[25] J. Canny, "A Computational Approach to Edge Detection," Readings in Computer Vision, pp. 184–203, 1987.