Cloud Computed Machine Learning Based Real-Time Litter Detection using Micro-UAV Surveillance

Ashley Chung
ahchung722@gmail.com

Sean Kim
seandyk@gmail.com

Ethan Kwok
ethan.c.kwok@gmail.com

Michael Ryan
michael_ryan_2000@yahoo.com

Erika Tan
eatan18@gmail.com

Ryan Gamadia*
ryangamadia96@gmail.com

New Jersey's Governor's School of Engineering and Technology
July 27, 2018

*Corresponding Author

***Abstract*—Litter can remain undetected and uncollected for extended periods of time, causing detrimental effects on the environment. Current solutions to mitigating these effects focus on severe legal action directed towards offenders or litter collection events. This paper proposes a micro-unmanned aerial vehicle (UAV) capable of real-time litter detection from video surveillance footage through an ensemble-based machine learning model. Performances of five different algorithms, two classifiers and three detectors, were compared to determine the strongest models to utilize in the ensemble method. The five models were trained in parallel with various public images of litter on the Google Cloud Computing Engine. Out of the two ensemble models tested, one being a custom-built ensemble and the other being a bagging ensemble, performance of the bagging ensemble demonstrates a significant improvement in performance over any individual model.**

I. Introduction

Undetected litter is a ubiquitous problem that has negative implications on quality of life, the environment, and the economy. When municipal solid waste is improperly disposed, it can become a hindrance not only in public areas, where they are most commonly found, but also in animal habitats. For example, litter containing harmful chemicals may contaminate the surrounding water and air, causing further habitat alteration by depleting levels of oxygen and light. In turn, this impedes the ability of habitats to support life, leading to a decline in species diversity [1]. However, not only does litter alter the Earth's environment, but it also directly affects the animals themselves. The ingestion and entanglement of various improperly disposed contaminants poses a considerable threat to wildlife in habitats around the world. Unfortunately, unless these biohazards are identified and removed, they will take many years to decompose and will continue to disrupt the ecosystem [2]. In addition, up to 90% of forest fires are started by another commonly disposed item: cigarettes [3]. These fires can not only cause damage to the forest ecosystem, but also cost millions of dollars in reparations.

Currently, most solutions to this problem are not automated; rather, they revolve around legislation or community efforts to manually remove litter. For instance, the punishment for minor infractions is a fine and an order to clean litter or complete community service. In New Jersey, first time offenders are fined between $100 to $500 and subjected to 20 to 40 hours community service [4]. Additionally, many organizations work very closely to maintain public areas. One of the largest organizations that upholds the sanitation of public areas is Keep America Beautiful. This organization founded the Great American Cleanup, the largest community improvement program in the US to renew and clean recreational areas, shorelines, and waterways by removing litter and debris [5]. Another solution implemented are anti-littering campaigns, such as Dont Mess with Texas, created by the Texas Department of Transportation [6][7]. While these solutions have impacted the issue of littering, they are not feasible long term solutions, as human surveillance is both dependent on human labor and time-consuming. Thus, a more effective long term solution would be an automated object detection system capable of detecting and locating litter solely from real-time transmission of micro-UAV surveillance footage.

II. Background

A. Micro-UAV

Micro-UAVs were chosen as an instrument to collect visual data because they are easily portable and inexpensive, especially when compared to drones of standard size. In addition, micro-UAVs have the ability to navigate in smaller and narrower spaces, which would be useful for areas such as urban environments. Because of their mobility, they can also monitor a larger territory than security cameras would otherwise cover. The drones that were used in this project are specifically Cheerson CX-10WD-TX models that are remote controlled over Wi-Fi and equipped with high-definition first-person view cameras. The utilization of the drones allows for real-time transmission of video and image data to be analyzed by the litter detection machine learning models.

B. Machine Learning and Computer Vision

This project combined machine learning and computer vision techniques to accomplish the automated detection of litter. Machine learning is a mathematical process that trains computers to gain intuition about certain data through the recognition of patterns and traits. Within the field of machine learning, computer vision aims to automate the understanding of visual data such as images and videos. It analyzes non-tabular data and extracts important information and patterns from these data. This project will train and test two types of models: classifiers and object detectors. Classification is defined as the categorization of an object into its respective class; for the purposes of this project, different types of litter will be classified in order to distinguish one type of waste from another. Detection categorizes the object in an image and also locates it by providing a bounding box around the object.

C. Classifiers

1) Convolutional Neural Network (CNN): CNNs are commonly used to perform object detection, group similar objects, and classify images. Similar to feature detectors, CNNs take multi-channeled images as inputs and apply mathematical functions known as kernels onto the image vectors in order to recognize the features that are deemed most important for each image. For certain algorithmic layers of the CNN, different functions are applied so that the features across the whole image are accounted for in different positions, a process known as convolution. Another process that CNNs utilize is pooling, in which the image is compressed by removing insignificant data. Convolution and pooling are repeated until a single dimensional array is produced. The network then classifies the image with a numeric confidence score. The backpropagation algorithm is then used to continually train the network and adjust the weights in the CNN layers. The goal of backpropagation is to minimize the prediction error of the CNN by adjusting the features and weights until an optimal combination of parameters is found. A sample error function that may be used with backpropagation is displayed below, in which the error function E of vectors y and y' computes the square of the Euclidean distance between the two vectors [8].

$$E(y, y') = \frac{\|y - y'\|^2}{2} \tag{1}$$

2) Support Vector Machine (SVM): Support Vector Machine is a machine learning classifier that separates data by constructing a hyperplane between categories. The SVM views each data point as a p-dimensional vector and draws a $(p-1)$-dimensional hyperplane. The mathematical representation of a SVM is written in Equation 2, with β as the weight vector and β_0 as the bias vector.

$$f(x) = \beta_0 + \beta^T x \tag{2}$$

From the infinite number of hyperplanes that exist between the categories, the SVM chooses the most optimal hyperplane that maximizes the margin, or twice the distance between the closest training data points to the hyperplane. The distance between a data point and a hyperplane is calculated as shown in Equation 3.

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

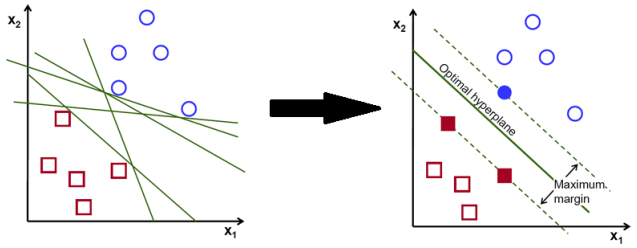


Fig. 1. Process of SVM drawing the optimal hyperplane [9]

To use the SVM as a classifier, the histogram of oriented gradients (HOG) was chosen as the feature descriptor, an algorithm that extracts features from an image, such as shape and texture [9]. The HOG feature descriptor calculates the distribution of direction of gradients, which is useful for determining the shape of an object [10]. Before calculations, the image is preprocessed to a certain size of choice before running the feature descriptor to maintain the same feature vector length for all images. Afterwards, the gradient values are calculated by applying a $[-1, 0, 1]$ derivative mask in both horizontal and vertical directions, which are represented by g_x and g_y. For each pixel, the gradient has a magnitude and direction, as shown in Equation 4 and 5.

$$g = \sqrt{g_x^2 + g_y^2} \tag{4}$$

$$\theta = tan^{-1}\left(\frac{g_y}{g_x}\right) \tag{5}$$

To calculate the histogram, the image is divided into cells of a certain size. Then, the gradients within this cell are stored by an array with a size defined by the number of orientation bins.

With this method, the feature descriptors are less resistant to noise and computationally effective. Finally, a block size is chosen to group multiple cells into a single element vector to store all the histograms for the cells in that block and normalized to be combined with other blocks [11][12].

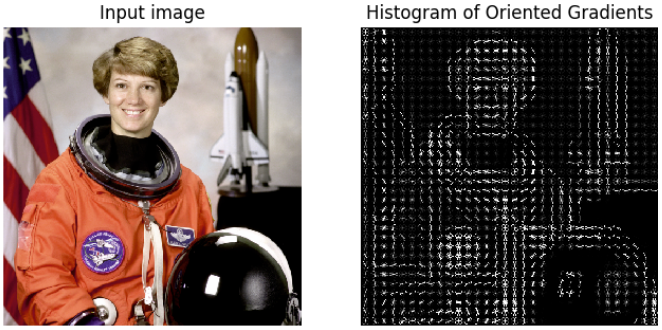


Fig. 2. Visualization of HOG [12]

D. Detectors

1) Single Shot Multibox Detector (SSD): Single Shot Multibox Detector (SSD) is a machine learning algorithm that detects objects by generating and finding a matching box with the ground truth box around an object in a single forward pass of the network. SSD incorporates a VGG16 CNN architecture with additional feature extraction layers. A ground truth box is a user inputed bounding box around an object in an image. First, SSD creates multiple bounding boxes, or k-anchors, in each cell on the pre-annotated image using the Multibox algorithm. Afterwards, as more convolutional layers are generated, it uses small convolutional filters to create multi-scale feature maps and thus computes the location and class scores of the object. Higher resolution feature maps are responsible for detecting small objects and, lower resolutions are used to detect large objects. The number of filters applied around each location in a feature map is calculated with the following equation: $(c+4)k$, where c is the number of classes and k is the number of bounding boxes at a given location.

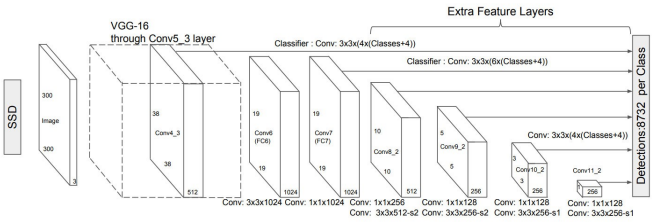


Fig. 3. Visual diagram of SSD [13]

SSD predictions are classified as positive matches or negative matches. The intersection over the union (IoU) is an evaluation metric used to determine positive and negative matches. If the IoU is greater than 0.5 between the predicted bounding box and the ground truth box, then the match is positive. Localization loss, the mismatch between the ground truth box and predicted box from the Multibox algorithm, is used to get positive matches closer to the ground truth box. The confidence loss is the loss in making a class prediction. For positive predictions, the loss is compared to the confidence score of the corresponding class. On the other hand, the loss for negative predictions is compared to the class where no objects are detected. The final loss function is computed as

$$L(x,c,l,g) = \frac{1}{N}(L_{conf}(x,c) + \alpha L_{loc}(x,l,g)) \qquad (6)$$

where N is the number of default matched bounding boxes, x is 1 if the default box matches the ground truth box and 0 otherwise, l is the predicted bounding box parameters, g is the ground truth bounding box parameters, and α is the weight for the localization loss [13].

2) Region-Based Fully Convolutional Network (R-FCN): R-FCNs seek to achieve accurate and quick predictions for image location by evaluating object locations using a minimal number of CNNs. In order to accomplish this, the R-FCN runs a fully convolutional region proposal network to detect regions of interest where the model should look more thoroughly for significant objects. Next, a CNN runs on the entire image to detect important features.

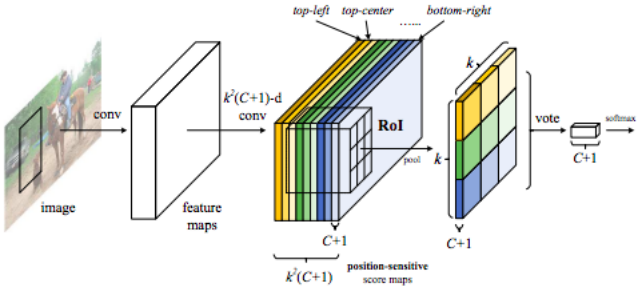


Fig. 4. R-FCN Architecture [13]

Since the convolutional neural network learns location invariance during training, it has no method of determining the position of objects of interest in the image. To resolve this issue, the model splits each region proposed by the region proposal network into k^2 subregions, where k is a selected constant value, and adds a fully convolutional layer to the end of the CNN with $k^2(C+1)$ filters. In this equation, C represents the number of classes that the R-FCN is trying to detect. The model trains these filters to specifically look for region based features on the object of interest. For example, one filter might train to look specifically for the upper left hand corner of a water bottle, while another filter might look for the center of the bottle. After checking the proposed region with the region based filters, the filters produce a score for how confident the proper part of the object is in the proper section of the proposed region. If the filters produce an overall high confidence score indicating that the region of interest contains the object in the proper position, then the model predicts that the object is in that region (See Figure 4) [14].

3) You Only Look Once (YOLO): You Only Look Once (YOLO) is another object detection machine learning algorithm that finds objects in an image by creating a series of bounding boxes and determining the box with the highest probability containing the object. While the YOLO network lags behind other object detection algorithms in accuracy, it is able to quickly detect objects, making it well suited for real time detection [15]. Unlike other algorithms, YOLO generates all its predictions at once, hence the name, as it only processes the image once. The algorithm first divides an image into a grid of S^2 squares where S is a constant value. If a square of the grid lies upon an object, then that square is responsible for creating B bounding boxes on the object. The bounding boxes also are paired with a confidence score. YOLO generates a confidence score by assessing how certain it is that a bounding box contains an object or part of an object. If there is no object in a bounding box, then the confidence score should be 0. Formally, the confidence score is defined as

$$Pr(Object) \times IOU_{\text{pred}}^{\text{truth}} \tag{7}$$

where $Pr(Object)$ is the probability of how confident YOLO thinks the bounding box is and the IOU is the intersection over union.

In words, the confidence score is equal to the IOU between the predicted and the ground truth box. After all bounding boxes are generated, YOLO looks at all bounding boxes and holistically determines how confident it is that a bounding box or a group of bounding boxes contains an object with the equation

$$\begin{aligned} &Pr(Class_i|Object) \times Pr(Object) \times IOU_{\text{pred}}^{\text{truth}} \\ &= Pr(Class_i) \times IOU_{\text{pred}}^{\text{truth}} \end{aligned} \tag{8}$$

where $Pr(Class_i)$ is the probability probability of of an object being of a certain class and $Pr(Class_i|Object)$ is the probability of of an object being of a certain class given the probability of how confident YOLO thinks the bounding box is.

This provides a class-specific confidence score for each bounding box. This overall score determines both the probability that a certain object appears in the box and how well the predicted bounding box fits the object. If the confidence score is higher than a certain threshold, then YOLO will finally propose that there is a certain object in a picture [16].

E. Ensemble Models

Individual machine learning methods can achieve a high degree of accuracy with enough training and optimization. One powerful method to boost the accuracy and strength of machine learning models even further is to use an ensemble method to combine multiple models together. In an ensemble method, various machine learning models individually make predictions on the image, and the predictions are combined together in order to generate a more accurate prediction. Common ensemble methods include stacking, bagging, and boosting. Stacking involves feeding the outputs of one model into the inputs of another model. In a stacking model, the first models act as feature descriptors for the later models which draw more sophisticated predictions, similar to adding layers in deep learning. Bagging involves averaging together the predictions of multiple algorithms using a weighted average where the highest performing models having the strongest weight in the prediction. Boosting performs in a similar manner to bagging, except a higher focus is placed on training models on images that other models failed to predict correctly instead of randomly distributing the training data. Of these three ensemble methods, both a simple bagging technique and a custom stacking algorithm was implemented [17].

F. Python

Python is a high-level object-oriented programming language with emphasis on code readability and libraries. This general-purpose programming language provides powerful implementations to facilitate large data and advanced calculations with libraries such as TensorFlow, Keras, Scikit-Learn, and NumPy. Tensorflow, Keras, and Scikit-Learn are libraries that provide various machine learning frameworks and algorithms, such as neural networks and support vector machines. NumPy is a library for applying various advanced mathematical functions with arrays and matrices, both key components of machine learning algorithms. To use computer vision, Python was also implemented with Scikit-Image and OpenCV. These libraries provide important image manipulation tools computer vision techniques, such as feature extraction and image classification.

G. Google Cloud Platform

The Google Cloud Platform encapsulates various tools for developers such as data analytics, machine learning, and data storage. A sub-tool of this service is the Google Compute Engine, which provides virtual machines running in Googles data centers and network. For this project, Google Cloud computing was implemented to efficiently train and test the algorithms. Because there was limited access to computers with powerful processing power, this often resulted in time-consuming training periods and poorly optimized results. The utilization of Google Compute Platform enhanced the training of models as its virtual machines provided greater graphics processing power, thus training the algorithms rapidly and efficiently [18][19].

III. Experimental Procedure

A. Dataset Preparation

The images used to train and test the models were pooled together from two main sources: Trashnet Dataset from Gary Thung and Mindy Yangs final project for Stanford University CS 229 Machine Learning course and various litter images from Google Images. The Trashnet dataset consisted of 2527 images of litter taken on a white background and resized to a 512x384 resolution. The images in the Trashnet Dataset were separated into six categories: glass, paper, cardboard, plastic, metal, and trash. (See Figure 5). After looking through the

Trashnet dataset, 2342 images were selected as useful for the final dataset based on the quality and contents of the pictures. To further supplement the dataset, the tool Google Images Download was utilized to gather a variety of litter images through downloading multiple images from a Google Image search result [20]. Similar to the Trashnet dataset, the downloaded images were checked for quality and content, resulting in 483 images. Supplementing the Trashnet dataset with images from Google Images helped add color to the background of the dataset because Trashnet consisted solely of images with a white backdrop. This ensured that the model would learn in a more versatile manner in litter detection rather than solely detecting litter on white backgrounds, which is not practical for implementing the model to be used in litter detection outside. After gathering all the images, the images were divided up into eight new labels, as seen above: bottle, can, cardboard, container, cup, paper, scrap and wrapper.

Fig. 5. Examples of images of litter from the Trashnet dataset used in constructing the final dataset: (a) bottle; (b) can; (c) cardboard; (d) container; (e) cup; (f) paper; (g) scrap; (h) wrapper

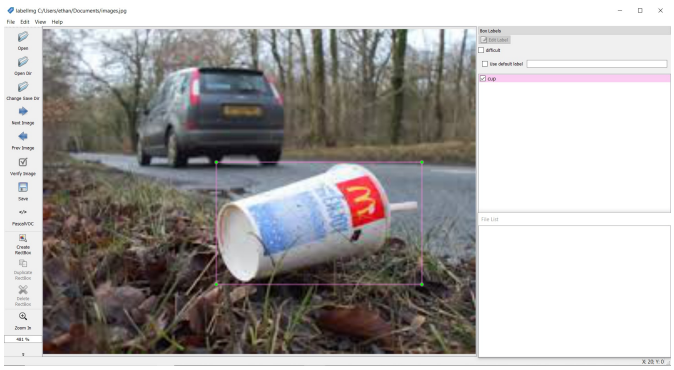


Fig. 6. Labelled image of a wrapper using LabelIMG

For classifier purposes, only images from the Trashnet dataset could be used. Due to the nature of classifiers used in this project, the dataset could only be consisted of images with one object. Furthermore, the images were further resized to 160x120 to decrease computation time while maintaining the aspect ratio. The Trashnet dataset was randomly split with a 80:20 training to testing ratio, resulting in 1873 training images and 469 testing images. On the other hand, the detectors were able to utilize the entire dataset. To generate the necessary annotations, the tool LabelImg was used to generate the annotations in an XML file using PASCAL visual object class (VOC) format [21]. The combined dataset was also randomly split with a 80:20 training to testing ratio, resulting in 2259 training images and 566 testing images. The exact distribution for each category can be seen in the table below. The number of images and number of objects are not equal for the combined dataset because some images from Google Images contained multiple pieces of litter.

TABLE I
DATASET SPECIFICATIONS

	Classifier (Trashnet Only)	Detector (Combined Dataset)
Bottle	0.681	0.647
Can	0.489	0.510
Cardboard	0.588	0.689
Container	0.442	0.419
Cup	0.000	0.889
Paper	0.000	0.746
Scrap	0.154	0.000
Wrapper	0.282	0.643
Average	0.330	0.623

B. Evaluation Methodology

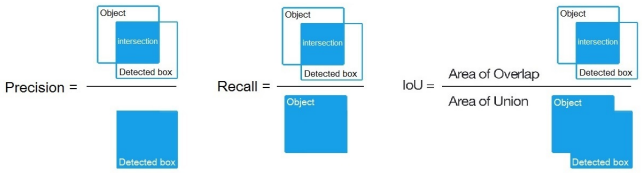


Fig. 7. Labelled image of a wrapper using LabelIMG

There are various methods to analyze the efficiency of a machine learning algorithm. For the object detectors, the mean average precision (mAP) was used. mAP is an evaluation metric that measures the accuracy of object detection algorithms. mAP uses intersection over union (IoU) to measure the overlap between a bounding box with the pre-labeled bounding box, also known as the ground truth box, and thus the accuracy [22][23]. If the IoU is greater than 50 percent, then the prediction is positive or correct. In the Equations (9) and (10), true positive (TP) means the number of detections with IoU greater than 50 percent. False positive (FP) means the number of detections with an IoU less than or equal to 50 percent or detected more than once. False negative (FN) means the number of objects not detected or was detected with IoU less than or equal to 50%. Precision measures the fraction of positive predictions that are correct by dividing the amount of correct positives found over the total amount of detections. Recall measures the fraction of correct positive predictions out of all actual positive entries. Average precision (AP) is calculated as the average of maximum precisions of all recall values for a specific class. mAP is the average of APs from all classes.

$$Precision = \frac{TP}{TP + FP} \tag{9}$$

$$Recall = \frac{TP}{TP + FN} \tag{10}$$

In order to evaluate the classifiers, namely CNN and SVM, a separate metric was used, as classifiers do not assign bounding boxes to objects. The chosen metric was the F1 score, which is the harmonic average of precision and recall [24]. The harmonic average is defined as the number of items being averaged divided by the reciprocal of each item.

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{11}$$

C. Google Cloud Implementation

In order to optimize the training time for each algorithm, two virtual machines were created. Each machine ran on Ubuntu 16.04 and was implemented with a Intel Haswell _quad-core central processor and two Nvidia Tesla P100 graphic processors. In order to interface with the Google virtual machine, an SSH client was connected to the virtual machine through Putty, a software that allows connection with remote computers. This granted a command prompt connection to the Ubuntu 16.04 OS and allowed software installation. To run the machine learning networks, libraries such as Scikit and Pytorch were installed. Keras, Anaconda, and TensorFlow 1.8 were also downloaded as frameworks to allow the training of the neural networks. The time spent on training was reduced even further by installing CUDA 9.2 and CUDNN 7.1.4, both of which allow the usage of graphic processor units through TensorFlow in the training. However, Google Compute Engine is set to compile with an older version of Tensorflow and CUDA. Using Bazel, the most recent versions of Tensorflow, CUDA, and CUDNN were built on the Ubuntu machine, increasing the processing speed by roughly 7 to 12% [25].

D. Overall Algorithm Structure

The process of building the final detector model for use in conjunction with the micro-UAV began by splitting up the dataset for labeling by individual team members. After the dataset was fully annotated and consolidated, the dataset was divided up again, this time into training and testing data. The five separated models were uploaded to a Google Cloud instance for training and the training data was fed to each of the five separate models. Evaluation metrics were assessed for each model using the testing data. At this point the two strongest detector models and strongest classifier were selected for use in the ensemble method to strengthen the accuracy of the prediction. These three models were joined together using a custom ensemble method to create one predictor model. Additionally, the two strongest detector models were selected for use with a Bagging ensemble method. These two ensemble methods were tested and compared to find the best final model for use with the micro-UAV footage. (See Appendix A)

E. CNN Implementation

The technique of transfer learning was used to further train the VGG16 and InceptionV3 base convolutional neural networks with four additional layers. VGG16 is a pre-trained deep learning network; similarly, InceptionV3 is a convolutional neural network that has been pre-trained on an image dataset made publicly available by Princeton and Stanford University called ImageNet and consists of convolution and max pooling layers. Because they are able to extract features from images, both are useful for image classification.

In order to fine-tune the VGG16 and InceptionV3 models, more layers were added. The layers that were added to VGG16 are shown below.

```
x = Flatten()(base_model.output)
x = Dense(500, activation='relu',
    name='fc1')(x)
x = Dropout(0.5)(x)
x = Dense(8, activation='softmax',
    name='fc2')(x)
model = Model(inputs=base_model.input,
    outputs=x)
```

The Flatten layer converted all of the pooled images taken from the dataset into a continuous one-dimensional vector. The Dense layers conduct a linear operation on the input vector that was given. Dropout prevented overfitting the neural network on the litter dataset through a random elimination of units and connections between layers; it essentially regularizes the neural network. Training both of the models revealed a better performance for VGG16; thus, it was used as the final CNN classifier.

F. SVM Implementation

The SVM model was mainly developed with Scikit-Learn and Scikit-Image. The training process started with the feature extraction from the training images. The HOG function from Scikit-Image extracted and saved the histograms from each image. An 8x8 cell, 1x1 block, and 12 orientation bins were used as parameters for the feature descriptor, resulting in a vector size of 3600 per image. These values were chosen to maintain the useful features while also saving computation time. After saving all the feature descriptors into separate files, the SVC module with a linear kernel from Scikit-Learn was used to train and save the necessary classifier. The testing process followed a similar structure; images in the the testing process were passed through the same feature extraction process as the training step.

G. SSD & R-FCN Implementation

The SSD and R-FCN models were built using the Tensorflow Object Detection API [26]. The R-FCN was built from the Tensorflow R-FCN Resnet-101 model and the SSD was built using the Tensorflow SSD InceptionV2 model. Both models were previously trained on the MSCOCO Dataset using over 220,000 annotated images. Transfer learning was used to instantiate the new R-FCN model with some of the previously trained layers of the Resnet-101 network and the SSD model with previously trained layers from Googles Inception v2. In order to use the Tensorflow Object Detection API the images and image annotations needed to be converted from PASCAL-VOC format to tfrecord files. A simple script was written in

python to convert the training and testing annotations into tfrecord files.

After preliminary testing with the models, it became clear that some of the hyperparameters needed improvement for both models. During training of the R-FCN model the loss value fluctuated instead of steadily decreasing. To mitigate this issue the learning rate of the model was decreased slowing the learning process, but making training more gradual and consistent. The rate of decrease of the learning rate was also slightly increased, so that the model would learn quickly at the beginning of the training process, but learning would slow down as the model progressed again to make changes more gradual. During training for the SSD model the loss value stopped decreasing after training the model for only a little bit. Similar changes were implemented to solve this problem with the SSD. The learning rate was changed from 0.0004 to 0.0001 and the learning rate decay was decreased from 0.95 to 0.9. After these small adjustments the models were ready for training on the final dataset.

H. YOLO Implementation

The YOLO neural network was mainly implemented with DarkFlow, an open source adaptation of the Darknet library for TensorFlow and Python [27]. Specifically, the algorithm was generated with the architecture and weights of Tiny YOLO, which is a more compressed version of YOLO trained with the MSCOCO dataset. In order to modify the YOLO architecture for the purposes of litter detection, the output layer was changed to have 8 possible litter classifications and 65 filters. Similar to other algorithms, the dataset of annotations was split into a training set and a testing set. Using a built-in function, 80% of the dataset was randomly assigned to training, while the other 20% was designated for testing.

Furthermore, the hyperparameters of YOLO were modified to attain more accurate results and a higher mAP score. Parameters that affect the YOLO algorithm include epochs, batch_size, stopbackward, and learning_rate. These variables greatly affect the results of the algorithm, often trading off bias with variance. After creating the implementation of the neural network, it was found that the network had a tendency to poorly detect smaller objects and generally needed to have a greater number of iterations. By changing the learning_rate and batch_size hyperparameters, the YOLO was able to converge to a global optimum. Through brute force, it was found that the YOLO model works best with a learning_rate of $1 * 10^{-5}$ and a batch size of 16.

I. Custom Ensemble Implementation

A custom ensemble method was designed and implemented in order to make use of both object detection and object classification algorithms to extrapolate a more accurate prediction for the bounding boxes generated by the models. The ensemble utilized two object detection algorithms and one object classification algorithm. The custom ensembler begins by running the first object detection model to propose its predictions for possible locations of litter in the image.

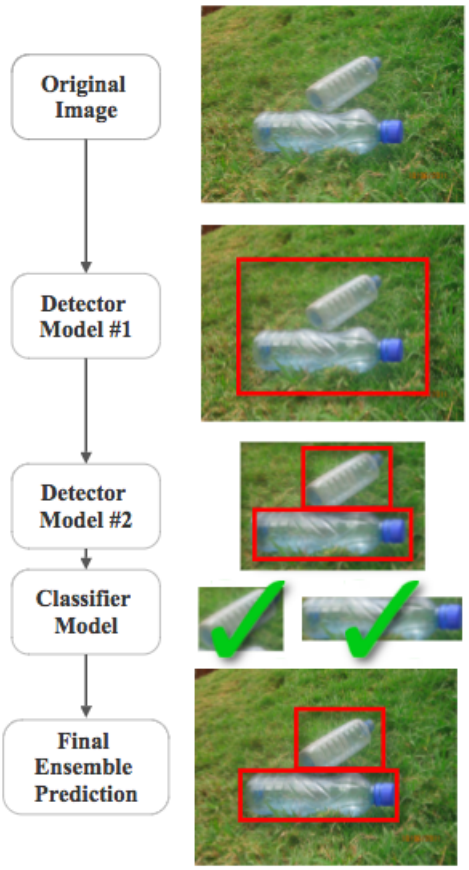


Fig. 8. Custom ensemble method architecture

The first object detection model acted as a region proposal network for the later models. This means that the model predicted the important regions in the image where objects of interest may be located. For this role an ideal model to select would be a quick model with high accuracy at detecting the general region a target object presides in, without spending too many computational resources shrinking the bounding box down. SSD was selected as the first detector, because it had the highest precision and accuracy score of all detector models, meaning it was the strongest model for picking out objects from the general image. In the next step, the custom ensemble method runs the second object detection model on each proposed subregion from the first model. In this way the second detector model serves to shrink down the predictions of the first detector model, or catch anything the first detector may have missed within the proposed subregions. For this role in the ensembler the ideal model would potentially sacrifice some speed for accuracy. This reduction in speed for higher accuracy should not cause much of an issue for the speed of the final model, because the slower detector only has to run on smaller segments of the image instead of the full image. R-FCN was selected for this role because of its fairly strong accuracy and precision relative to other models. The last step in the custom ensemble method involves running the classifier model on each subregion proposed by the second

detector model within the regions proposed by the first model to determine if the smallest predictions have merit or if the second detector simply shrunk the prediction too small. If all of the bounding boxes proposed by the second model match the classes assigned by the classifier, then the predictions of the second model are selected in place of the predictions of the first model. Otherwise, the predictions of the first model are accepted. In this way the various models serve to check the predictions of the other models to produce a more accurate ensemble (See Figure 8).

J. Bagging Ensemble Implementation

The bagging ensemble was built with the two best performing detectors. Predictions from each detector were averaged together to determine the final prediction. Due to the possibility of detecting multiple objects in one image, the ensemble method needed to determine which bounding boxes corresponded to the same object. This step was critical to ensure the ensemble would not ruin predictions with multiple objects by averaging the box to a point in the image that likely does not contain any object at all. In order to match the boxes together, the IoU values were generated for every combination of boxes. The bounding boxes were considered to be correlated and were averaged together only if the IoU score exceeded 50%. Otherwise the predictions were considered separate entities. If a bounding box did not have any other boxes correlated to it, the prediction would be added to the final prediction only if the original model had an exceedingly high confidence value [28].

K. Micro-UAV Integration

Connecting the micro-UAV to a computer in order to stream live footage presented a unique challenge in the project. The micro-drone came with an app by which the device could be controlled from a smartphone; thus, it was known that the drone had some level of a communication protocol. However, the software for controlling the micro-UAV was not publicly available, thus an open-source project named Hack-a-Drone was implemented to control and receive video stream from the drone directly to a computer [29]. The buffered image files were piped from the project, which was written in Java, to a separate Python script through a localhost socket in order to make it compatible with the machine learning models which were also written in Python. Although it sacrificed a small amount of the speed of the real time detection since the connection from the drone to the Python script was less direct, this method saved time in implementation since the drone communication protocol did not need to be programmed from scratch. In order to minimize the delay in the video feed, a multithreading approach was utilized to capture drone footage in one thread while processing the footage in another. With the real time video feed connected from the drone, the model could finally be tested in a working demonstration.

IV. Results and Analysis

A. CNN

While the CNN was able to achieve a low training loss of 0.1598, the average F1 score of the test set showed that the model had clearly overfitted the training dataset. Two methods were brought in to relieve overfitting from the model: the creation of a validation subset and the increase of the learning rate. Hyperparameter tuning resulted in a model that demonstrated a noticeable improvement, with an accuracy of 51.73% and an average F1 score of 0.330. The F1 scores of 0.0 regarding the cup and scrap classes may have resulted from the fact that there was an unequal distribution of image data. There were fewer images of cups and scraps to train on compared to other types of litter such as bottles; thus, there was not as much information that the models could learn from. As a result, the model rarely classified litter as a cup or a scrap.

B. SVM

As summarized in Table II, the SVM achieved an F1 score of 0.623. The scrap class particularly performed poorly with a F1 score of 0. Various parameters of the feature descriptor were modified to achieve the best score possible. Similar to the CNN, the SVM faced setbacks due to the lack of images. With the skewed data, the SVM faced difficulties choosing the most optimal hyperplane. Due to the SVM dividing data into two different groups, the SVM faced some difficulties in classifying the images into eight different categories.

TABLE II
F_1 Scores of Classifiers

	CNN	SVM
Bottle	0.681	0.647
Can	0.489	0.510
Cardboard	0.588	0.689
Container	0.442	0.419
Cup	0.000	0.889
Paper	0.000	0.746
Scrap	0.154	0.000
Wrapper	0.282	0.643
Average	0.330	0.623

C. SSD

SSD trained for longer than the other two detection algorithms (R-FCN and YOLO) but it had the greatest mAP score. The bottles, cans, containers, paper, and cardboard classes had the greatest amount of images, so the algorithm was able to be trained more thoroughly. The cup class had a lower accuracy score because there was less than 100 images for cups and, thus there was not as much data to learn from. Overall, the SSD accurately detected objects over 50% of the time.

D. R-FCN

The R-FCN performed well with an overall mAP of 0.525. Although it was outperformed by SSD, the difference between the mean AP scores of the two models was only about 0.025, suggesting a similar performance level. R-FCNs highest

accuracy scores were seen in the paper, cardboard, and bottle classes. This is consistent with the fact that there were a higher number of images for those three classes compared to others. The R-FCN did not perform as well during its first training process. Although the mAP score of this initial model was never assessed, the model had difficulty detecting any litter reliably. Between training sessions, the learning rate was decreased in order to fine-tune the model, which significantly boosted its performance.

E. YOLO

The tiny YOLO algorithm was successfully trained on the Google Cloud Platform with the entire dataset of 2826 images. However, tiny YOLO performed very poorly with an overall average mAP score of 0.0825. The second YOLO algorithm used a more robust version of YOLO that contained more convolutional layers, allowing it to make more accurate predictions. As a result, the average mAP score improved by 0.2605.

The YOLO model performed with an average AP score of 0.404, which was significantly less compared to the other two object detection algorithms. This may be attributed to the fact that YOLO is a notably faster detection algorithm, and therefore greatly trades off its accuracy for speed. YOLO also had a considerable demand for data, as the neural network performed very poorly on the wrapper, cup, and scrap classes, all of which had roughly less than 100 images each. In addition, due to its methodology, YOLO generally is known to perform worse on smaller objects. This issue was undoubtedly present in the same aforementioned classes, thus significantly weakening YOLO's average mAP score.

TABLE III
AP SCORES OF DETECTORS

	SSD	R-FCN	YOLO
Bottle	0.62	0.71	0.56
Can	0.70	0.55	0.48
Cardboard	0.52	0.73	**0.79**
Container	0.71	0.68	0.55
Cup	**0.12**	0.00	0.00
Paper	0.80	0.74	0.84
Scrap	**0.67**	0.56	0.00
Wrapper	0.32	0.22	0.00
Average	0.56	0.53	0.40

F. Custom Ensemble Results

The custom stacking ensemble method was built successfully by consolidating the three best performing models and stacking them together into one model. It was determined that the version of the ensemble with the SSD feeding into R-FCN feeding into the SVM had the most favorable average mAP score of 0.521. While the algorithm performed with a small deviation of 0.0048 from the R-FCN, it was still outperformed by SSDs score of 0.5569. This may be because the second level detector focused too closely on an object causing the classifier to inaccurately recognize an object from a very miniscule bounding box. The SVM classifier also only had an F1 score of 0.623, which was still not robust enough to be the final classifier on the ensemble method. Furthermore, the Trashnet dataset contained many images of only specific parts of litter, such as only the upper half of a water bottle. This may have confused the model when shrinking the image, as the second detector model often split the object into pieces and the classifier would still detect each piece as the correct object of interest. For these reasons, the checks and balances system between the models in the custom ensemble method did not improve on the sum of its parts.

G. Bagging Ensemble Results

The Bagging Ensemble method performed better than the individual detector algorithms with an mAP score of 0.6434. This was around 8.6% better than the best individual model, the SSD. This ensemble method likely performed better than the custom ensembler because through averaging the results of the strong detectors, the ensemble method balanced inaccuracies in the predictions of individual models. The ensemble method also made fewer false positive predictions. This is because the boxes produced by each individual model needed extremely high confidence scores to overwrite the other model if both models did not agree. The bagging ensemble method demonstrated the strongest results of all other ensemble methods and individual models because of the measures it took to ensure high quality predictions. Since the bagging model had the best performance, it was selected for processing the real-time video from the micro-UAV.

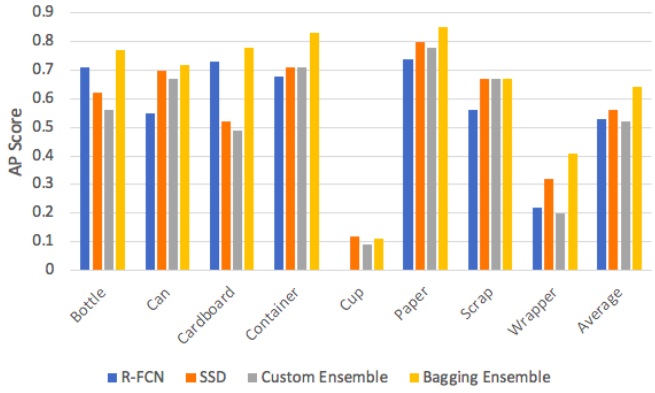


Fig. 9. Comparison of R-FCN, SSD, Custom Ensemble, Bagging Ensemble

V. CONCLUSION

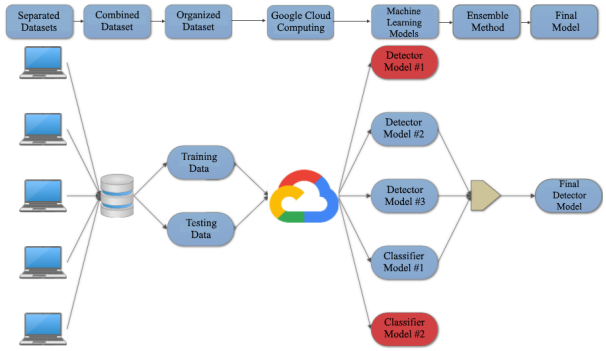
The use of robust machine learning algorithms alongside with computer vision techniques has a great potential to affect how the environment is monitored and maintained. It was discovered that while the complex custom ensemble did not create any noticeable improvements compared to the robust object detection algorithms individually, the more simplistic bagging ensemble was able to significantly improve the accuracy of the combined strong models. When tested, the bagging ensemble was able to perform real-time litter detection from the drones

video stream with the highest accuracy. Clearly, the key to improving computer vision and object detection algorithms is to create ensembles from multiple diverse models. More training images, especially of scrap and wrappers, could be used to further train and enhance the models. A larger dataset would also allow for the detection of a greater number of litter categories, permitting the identification unique brands of litter. Furthermore, a UAV with a higher-resolution camera and faster video transmission speed would allow for a more accurate and faster analysis of footage.

If research of this project is continued, the litter detection system could be implemented into real-world applications by deploying drones to public areas such as national parks and highways. With the installation of a GPS, the drones could report the precise location and type of litter to organizations to help communities stay clean. Additionally, the drones could be programmed with autonomous flight features to easily navigate around and find litter without human supervision. To further prevent littering, a mobile application using augmented reality could be used to reward users who pick up litter, providing a further incentive to pick up litter and clean the environment.

APPENDIX

Overall Flowchart of Final Model



ACKNOWLEDGMENT

The authors of this paper would like to acknowledge project mentor Ryan Gamadia and residential teaching assistant (RTA) Pragya Hooda for their guidance and involvement throughout the research process, in addition to Andrew Page and Google for their contribution of Google Cloud Platform credits. Moreover, special gratitude towards program director Ilene Rosen, associate program director Jean Patrick Antoine, research coordinator Brian Lai, and head RTA Nicholas Ferraro for making it possible to conduct high level research at the New Jersey Governors School of Engineering and Technology (NJ GSET). Special thanks to Anthony Yang, Shantanu Laghate, and Jennifer He for supplying preliminary resources and aid. Finally, much gratitude to the sponsors of NJ GSET for their continued participation and support: Rutgers University, Rutgers School of Engineering, the State of New Jersey, Silverline Windows, Lockheed Martin, Rubiks, and the alumni of NJ GSET.

REFERENCES

[1] Impacts of Mismanaged Trash, *EPA*, 23-May-2017. [Online]. Available: https://www.epa.gov/trash-free-waters/impacts-mismanaged-trash.

[2] R. Ahmad, Detrimental Effects of Littering, *EcoMENA*, 15-May-2018. [Online]. Available: https://www.ecomena.org/littering/.

[3] Wildfire Causes and Evaluations (U.S. National Park Service), *National Parks Service*. [Online]. Available: https://www.nps.gov/articles/wildfire-causes-and-evaluations.htm.

[4] States with Littering Penalties, *National Conference of State Legislatures*. [Online]. Available: http://www.ncsl.org/research/environment-and-natural-resources/states-with-littering-penalties.aspx.

[5] What We Do, *Keep America Beautiful*. [Online]. Available: https://www.kab.org/about-us/what-we-do.

[6] *Don't mess with Texas*. [Online]. Available: http://www.dontmesswithtexas.org/about/.

[7] K. Nodjimbadem, The Trashy Beginnings of Don't Mess With Texas, *Smithsonian.com*, 10-Mar-2017. [Online]. Available: https://www.smithsonianmag.com/history/trashy-beginnings-dont-mess-texas-180962490/.

[8] Y. LeCun, C. Cortes, and C. Burges, LeNet-5, Convolutional Neural Networks, *Yann LeCun*. [Online]. Available: http://yann.lecun.com/exdb/lenet/.

[9] R. Gandhi, Support Vector Machine - Introduction to Machine Learning Algorithms, *Towards Data Science*, 07-Jun-2018. [Online]. Available: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

[10] N. Dalal and B. Triggs, *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1. Washington DC, USA: IEEE Computer Society, 2005.

[11] C. Cortes and V. Vapnik, Support-Vector Networks, *Machine Learning*, vol. 20, pp. 273297, Sep. 1995.

[12] Histogram of Oriented Gradients,. [Online]. Available: http://scikit-image.org/docs/dev/auto_examples/features_detection/plot_hog.html#sphx-glr-auto-examples-features-detection-plot-hog-py.

[13] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, SSD: Single Shot MultiBox Detector, *arXiv*, vol. 5, Dec. 2016.

[14] J. Dai, Y. Li, K. He, and J. Sun, R-FCN: Object Detection via Region-based Fully Convolutional Networks, *arXiv*, vol. 2, Jun. 2016.

[15] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy, Speed/accuracy trade-offs for modern convolutional object detectors, *arXiv*, vol. 1, Nov. 2016.

[16] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, You Only Look Once: Unified, Real-Time Object Detection, *arXiv*, vol. 5, May 2016.

[17] D. Opitz and R. Maclin, Popular Ensemble Methods: An Empirical Study, *Journal of Artificial Intelligence Research*, vol. 11, pp. 169198, Jul. 1999.

[18] What is cloud computing? — Google Cloud, *Google*. [Online]. Available: https://cloud.google.com/what-is-cloud-computing/.

[19] Compute Engine - IaaS — Google Cloud, Google. [Online]. Available: https://cloud.google.com/compute/.

[20] Vasa, H., Google Images Download, (2015), GitHub repository, https://github.com/hardikvasa/google-images-download

[21] Tzutalin, D., LabelImg,(2015), GitHub repository, https://github.com/tzutalin/labelImg

[22] How to calculate mAP for detection task for the PASCAL VOC Challenge?, *Data Science Stack Exchange*. [Online]. Available: https://datascience.stackexchange.com/questions/25119/how-to-calculate-map-for-detection-task-for-the-pascal-voc-challenge.

[23] P. Henderson and V. Ferrari, End-to-end training of object class detectors for mean average precision, *ACCV*, 2016.

[24] Y. Sasaki, The truth of the F-measure , *Teach Tutor Mater*, Oct. 2007.

[25] Benchmark CIFAR10 on Tensorflow 1.8 with CUDA 9.2 on cifar10, *Python 3.6*, 31-May-2018. [Online]. Available: http://www.python36.com/benchmark-tensorflow-on-cifar10/.

[26] Kaiser, L., Tensorflow Models, (2016), Github repository, https://github.com/tensorflow/models/

[27] Redmon, J. (2018). Darknet: Open Source Neural Networks in C. [online] Pjreddie.com. Available at: https://pjreddie.com/darknet/

[28] Arhnbom, M., Ensemble-ObjDet, (2017), GitHub repository, https://github.com/ahrnbom/ensemble-objdet

[29] Berlin, N. Hutting, J. and Runge R., Hack-a-Drone, (2017), GitHub repository, https://github.com/Ordina-JTech/hack-a-drone/