CMPT 733

Prioritizing Aid from Above

Jillian Anderson, Brian Gerspacher, Brie Hoffman

Motivation and Background

Fruit trees such as coconuts and bananas are an important source of food and income for communities in the South Pacific. These same communities are at high risk for natural disasters such as hurricanes, tsunamis, and volcanic eruptions [1]. After such events, aid organizations collect imagery via unmanned aerial vehicles (UAV) and a human manually analyzes the images to assess damage to affected areas. This helps guide efficient distribution of aid, prioritizing communities whose food sources have been most affected. However, manual analysis of this data can take days to complete. This delay hinders organizations from responding to crisis efficiently.
In response, the World Bank, Open Aerial Maps and WeRobotics have launched the Open AI Challenge, focused on developing machine learning models to accelerate aerial image analysis surrounding humanitarian disasters [2]. This challenge was brought to our attention by Dr. Patrick Meier of We Robotics, via our CMPT 733 professor, Dr. Jiannan Wang.

Problem Statement

Given a single aerial image covering 3 square kilometers and a shapefile containing latitude and longitude coordinates for ~15000 coconut trees, banana trees, mango trees, papaya trees, power poles and settlements, train a neural network to detect the presence of such objects in similar high-resolution aerial images. This is a challenging problem because:

- Generally, object detection is more difficult than image classification because the network doesn't know beforehand where objects will be in the image, and so needs to perform the task of classification on many different regions of the image.
- Aerial imagery differs significantly from the images found in common image collections such as ImageNet and the COCO dataset. Therefore, transfer learning using weights from networks that were pretrained on such image collections is not as helpful as it usually is [3].
- Low variability in our training data limits the generalizability of our model.

Data Science Pipeline

Data Collection

The training data and aerial imagery provided by the Open AI Challenge were obtained via direct download[1]. The provided image covered an area of 1.6 km x 2.2 km with a resolution of 9 cm per pixel, and was centred over the village of Kolovai in the Kingdom of Tonga. Ground truth labels were provided in a shapefile which contained points describing the latitude and longitude of trees in and around the area captured in the image. Additional shapefiles were available containing labels for roads and buildings but we did not use them in this project. The labels in the training data were developed by the Humanitarian OpenStreetMap Team.

Data cleaning

To make the training data easier to work with we used the GeoPandas python library to convert the shapefile containing training data to GeoJSON format. We then used a Python script to convert the latitude/longitude values to pixel values within the image.

---

[1] https://export.hotosm.org/en/v3/exports/8a5ba924-1f34-4ed8-a4f6-7b0e2921c06e
https://drive.google.com/file/d/1rumWHzO3_CO40uXhaP69roUyfFzYCe20
https://map.openaerialmap.org/#/-175.34221936224426,-21.095929709180027,15/square/20002233030/5a28640ebac48e5b1c58a81d?_k=4yyxj6

Exploratory Data Analysis

We used the Pillow Python package to plot the coordinates specified with the ground truth labels onto the image and found they didn't line up with objects in the images. In the end we needed to do a coordinate transformation: Longitude values needed to be 'stretched', i.e. x-values were shifted away from the centre of the image by amounts weighted by distance from the centre of the image, and latitude values were shifted up by amounts that increased from the bottom to the top of the image. After performing this transformation, the training labels matched objects in the training image exactly. Another aspect of our EDA was to determine the class distribution of the dataset. We found that the labels were from seven classes: coconut, banana, mango, and papaya trees, power poles, settlements, and a class labeled 'other', with coconut trees outnumbering the second largest class, banana trees, by a factor of 4:1. The exact class distribution in our data set is shown in Table 1 of the Appendix.

Data integration

To prepare the image and labels for use in training a CNN, we again used Pillow to tile the TIFF image into 1260 smaller JPEGs.
For each resulting tile, we created an accompanying .txt file that contained a line for each object of interest that appeared in the image. Each line had an integer value representing the class, and four floating point values representing a bounding box.

Analysis/Modeling

For a description of the development of our CNN, see *Methodology*.

Visualizations

As described above, we used the Pillow Python package to visualize our training data as part of EDA. This allowed us to understand and devise the coordinate transformation necessary to use the data. We used Matplotlib for the rest of our visualizations. All Figures are in the Appendix.

- We monitored training sessions by plotting how the training error decreased. See Figure 1.
- In our process to iteratively unfreeze layers (see *Transfer Learning*), we did a line plot of the resulting models to identify the ideal number of layers to unfreeze. See Figure 2.
- We compared models by plotting bar graphs and line graphs of their average precisions. See Figure 3.

Methodology

Train/Test Split

We split our 1260 images randomly into sets of 1000 and 260 for training and testing, respectively.

Tool selection

To develop a CNN for tree detection, we chose to use Darknet, an open source neural network framework written in C and CUDA, along with the YOLO (You Only Look Once) CNN architecture developed by the creators of Darknet (https://github.com/pjreddie/darknet), specifically YOLOv2 [5]. Unlike sliding window or region proposal methods which are slow and hard to optimize, YOLO uses a single CNN to simultaneously predict bounding boxes and class probabilities for those boxes over an entire image [4].

We chose Darknet/YOLO because of its speed and ability to be used with real-time video. We envision our tool one day being used in real-time, such as on drones flying over disaster areas. YOLO's accuracy is on par with other SSD variants and it is 3x faster [5]. Lastly, YOLO is known to be generalizable to different image sizes and resolutions because it performs random resizes as part of training [4]. We hope that this strength will help combat the low variability in our training set.
See Table 2 for a description of the layers in the YOLOv2 architecture. Redmon and colleagues provide a full description of the YOLO model, including loss function and optimization methods in their papers You Only Look Once: Unified, Real-Time Object Detection [4] and Yolo9000: better, faster, stronger [5].

Seeking Guidance

We reached out to Dr. Greg Mori at SFU in early March because he is a computer vision expert and has previously worked with Dr. Patrick Meier doing object detection in aerial imagery. Through Dr. Mori we connected with one of his graduate students, Nelson Nauata. Our meeting with Nelson in late March was extremely valuable as he was able to clarify a number of computer vision concepts for us, and he provided a number of solid avenues of exploration for improving our model.

Model Tuning

Data augmentation: YOLO has built-in data augmentation. As part of training, random crops, rotations, hue, saturation, and exposure shifts are performed [5].

Class composition: In our initial efforts to get a working model, we tried training a model to detect just coconuts. Then we added bananas. Then we did all seven classes. We found that when we tried to get detections of all seven classes, we were still only getting detections of coconut trees and bananas. We believe this is due to having very few examples of the the other five classes in the training set. Our best results were always for coconut trees.
To correct the class imbalance we ran a training session where any images that contained only coconuts were excluded. This resulted in the training set being reduced by roughly one half.

Learning rate: We modified the learning rate and how it changed throughout training mostly as a means of stabilizing our training sessions, preventing exploding gradients. We didn't find that it improved our results.

Transfer learning: Following common wisdom, we applied transfer learning whereby we initialized training with a set of weights pre-trained on ImageNet and used training to learn only the final convolutional layer. Predictions made by the resulting weights were terrible, as seen in Figure 3. We then tried using the same pre-trained weights to initialize training but froze the first 13 (out of 23) convolutional layers and left the last 10 to learn. Predictions made by this model were much better (see Figure 3). We also tried a process where we supplied pretrained weights and trained just the final layer. Then passed the resulting weights to another training session, this time with the last two layers unfrozen. Then used those weights to train the last three layers, and so on. Figure 2 of the Appendix shows how the mAP values increased as we unfroze layers throughout this iterative process.

Anchor boxes: YOLO predicts bounding boxes by predicting offsets to anchor boxes provided as part of the training configuration. As suggested by Nelson Nauata, we investigated using k-means clustering on the bounding boxes in our dataset to produce anchor boxes that accurately reflected our training set. Ultimately this didn't prove worthwhile because we already knew that all of the objects in any of our classes all had the same height and width. We then tried providing these dimensions as initial anchor boxes but this didn't improve our results.

Evaluation of models: After performing hyper-parameter tuning based on training data, we evaluated our models using the testing set of 200 tiles held out from training. Mean Average Precision (mAP), a widely used evaluation metric for object detection systems, was used to evaluate these models [6]. Average precision (AP) is the average value of precision over all recall values. mAP is the mean AP value over all classes. Since we want precision to remain high even as recall improves, larger mAP values indicate better performance. A detection was considered a true positive if it had an intersection over union (IoU) of at least 0.5 with a ground-truth bounding box [6]. Otherwise, the detection was a false positive. In the case where a single ground truth bounding box has more than one detection overlapping with it, only one is considered a true positive.

Based on the mAPs (see Figure 3 in the Appendix), we chose the 'No Pretraining' model for use in our python and web applications. This model provided the highest mAP (0.52) of the models which used data augmentation, an important feature as it improves the model's generalizability. Other models of note include the transfer learning model with 13 frozen layers, and the model detecting seven classes, which both obtained the moderately high mAPs of 0.47. Transfer learning models take much less time to train than models trained from scratch, thus considering its high mAP may be a better choice when faced with time and computational constraints. The model detecting seven classes continued to perform on banana and coconut tree detections. This is promising, as the additional functionality of detecting objects such as papaya trees or power poles may be important in other situations.

Evaluation

Our best models obtained comparable mAPs to those achieved by other YOLOv2, Faster R-CNNs and other single shot detectors on the PASCAL VOC 2007 dataset (0.6-0.8) [5]. This gives us confidence in our model and its performance.

Additionally, our model is able to detect trees unlabelled in ground-truth data. Each of these detections are considered as false positives. This artificially lowers the mAP scores obtained by our models. Figure 5 in the Appendix shows an example tile for which our CNN identified 23 coconut trees, while only 6 were labeled in the ground truth. This finding further bolsters our confidence in our model.

There are still questions regarding whether our model will be successful working with images taken under different weather conditions, different terrain, etc given that all training images were cropped from a single image. We found aerial images of coconut trees online that were unrelated to our data set and our model was unable to make any detections in them (Figure 6 in the Appendix).The shape of the coconut trees as well as the background are quite different from our training images. We suspect that our model will perform similarly to how it performed on our test images, provided future images are taken by the UAVs that took our training image, in the region our training image was taken in. For an example of predictions made by our CNN, see Figure 4 of the Appendix.

In addition, we recommend that further work on this project should focus on improving the training data by adding new data taken under a variety of conditions.

Data Product

We built a Python application that invokes Darknet to generate predictions for a given image. It uses the Flask web framework to allow the image to be selected and predictions to be viewed in a browser. The application is available at http://ec2-35-161-243-71.us-west-2.compute.amazonaws.com:5000. The AWS Free Tier EC2 instance that we are using has limited computing power. Predictions are

slow due to a lack of a GPU. More powerful hardware would provide improved performance. But sample results have been saved and can be viewed without waiting through a lengthy computation.

Lessons Learned

- How to work with images in Python.
- Learned about CNNs and how they are used for object detection and image classification.
- How to use the Darknet framework.
- Learned the value of computing resources and managing long-running processes in deep learning.
- Learned the concept of mean average precision as a metric for evaluation of multi-class object detection systems.
- Learned about the importance of domain knowledge and attention to detail when it comes to data set preparation.
- Learned the importance of documenting all the things we tried to improve performance.
- How to write a web application in Python.

Summary

Our project aims to use computer vision and machine learning to automatically assess the damage caused by cyclones in the South Pacific. By training a convolutional neural network to detect and count different kinds of trees present in aerial images, we seek to improve the ability of aid organizations to respond efficiently in the immediate aftermath of a natural disaster. Training data was provided as part of the challenge and we used it to train an object detection system using the Darknet framework and the YOLOv2 CNN architecture. We trained and tuned our models using the GPUs on the computers in ASB10928 throughout March and early April 2018, evaluating our results using the metric of mean average precision (mAP). In the end, our best results achieved a mAP of 0.52 using a trained from scratch model. This model was integrated into a user-facing web application. These results were submitted to Patrick Meier on April 16 as part of WeRobotic's Open AI Challenge.

References

1. Noy, I. (2016). Natural disasters in the Pacific Island Countries: new measurements of impacts. *Natural Hazards*, *84*(1), 7-18.
2. Open AI Challenge: Aerial Imagery of South Pacific Islands. Patrick Meier. Retrieved April 12, 2018, from https://werobotics.org/blog/2018/01/10/open-ai-challenge/
3. Radovic, M., Adarkwa, O., & Wang, Q. (2017). Object Recognition in Aerial Images Using Convolutional Neural Networks. *Journal of Imaging*, *3*(2), 21.
4. Redmon, J., Divvala, S., Girshick, R., Farhadi, A. You only look once: Unified, real-time object detection. arXiv preprint arXiv:1506.02640, 2015.
5. Redmon, J., Farhadi, A. Yolo9000: better, faster, stronger. *arXiv preprint arXiv:161208242* 2016.
6. Zhong, Y., Han, X., & Zhang, L. (2018). Multi-class geospatial object detection based on a position-sensitive balancing framework for high spatial resolution remote sensing imagery. *ISPRS Journal of Photogrammetry and Remote Sensing*, *138*, 281-294.

Appendix

Table 1: Class Distribution

Coconut Trees	10315
Banana Trees	2729
Mango Trees	261
Papaya Trees	97
Power Poles	112
Other	142
Settlement	5

Table 2: CNN architecture for detection with YOLOv2

Layer	Type	Filters	Size/Stride	Input	Output
0	Convolutional	32	3x3	416x416x3	416x416x32
1	Maxpool		2x2/2	416x416x32	208x208x32
2	Convolutional	64	3x3	208x208x32	208x208x64
3	Maxpool		2x2/2	208x208x64	104x104x64
4	Convolutional	128	3x3	104x104x64	104x104x128
5	Convolutional	64	1x1	104x104x128	104x104x64
6	Convolutional	128	3x3	104x104x64	104x104x128
7	Maxpool		2x2/2	104x104x128	52x52x128
8	Convolutional	256	3x3	52x52x128	52x52x256
9	Convolutional	128	1x1	52x52x256	52x52x128
10	Convolutional	256	3x3	52x52x128	52x52x256
11	Maxpool		2x2/2	52x52x256	26x26x256
12	Convolutional	512	3x3	26x26x256	26x26x512
13	Convolutional	256	1x1	26x26x512	26x26x256
14	Convolutional	512	3x3	26x26x256	26x26x512
15	Convolutional	256	11	26x26x512	26x26x256
16	Convolutional	512	3x3	26x26x256	26x26x512
17	Maxpool		2x2/2	26x26x512	13x13x512
18	Convolutional	1024	3x3	13x13x512	13x13x1024
19	Convolutional	512	1x1	13x13x1024	13x13x512
20	Convolutional	1024	3x3	13x13x512	13x13x1024
21	Convolutional	512	1x1	13x13x1024	13x13x512
22	Convolutional	1024	3x3	13x13x512	13x13x1024
23	Convolutional	1024	3x3	13x13x1024	13x13x1024
24	Convolutional	1024	3x3	13x13x1024	13x13x1024
25	Route	16			
26	Convolutional	64	1x1	26x26x512	26x26x64
27	Reorg			26x26x64	13x13x256
28	Route	27 24			
29	Convolutional	1024	3x3	13x13x1280	13x13x1024
30	Convolutional	(num classes + 5)*5	1x1	13x13x1024	13x13x35
31	Detection				

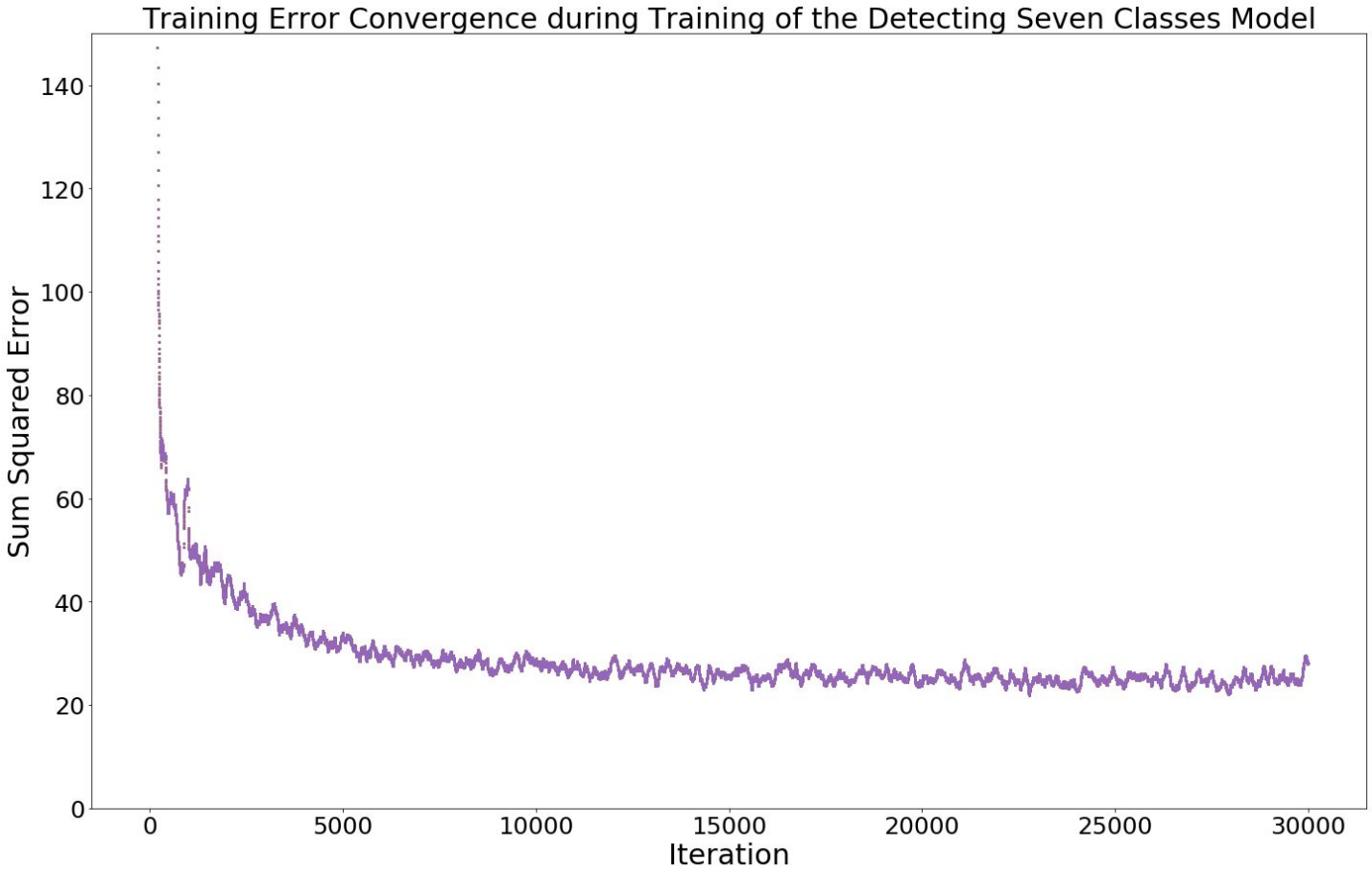

Figure 1: An example plot of how training error decreased during training

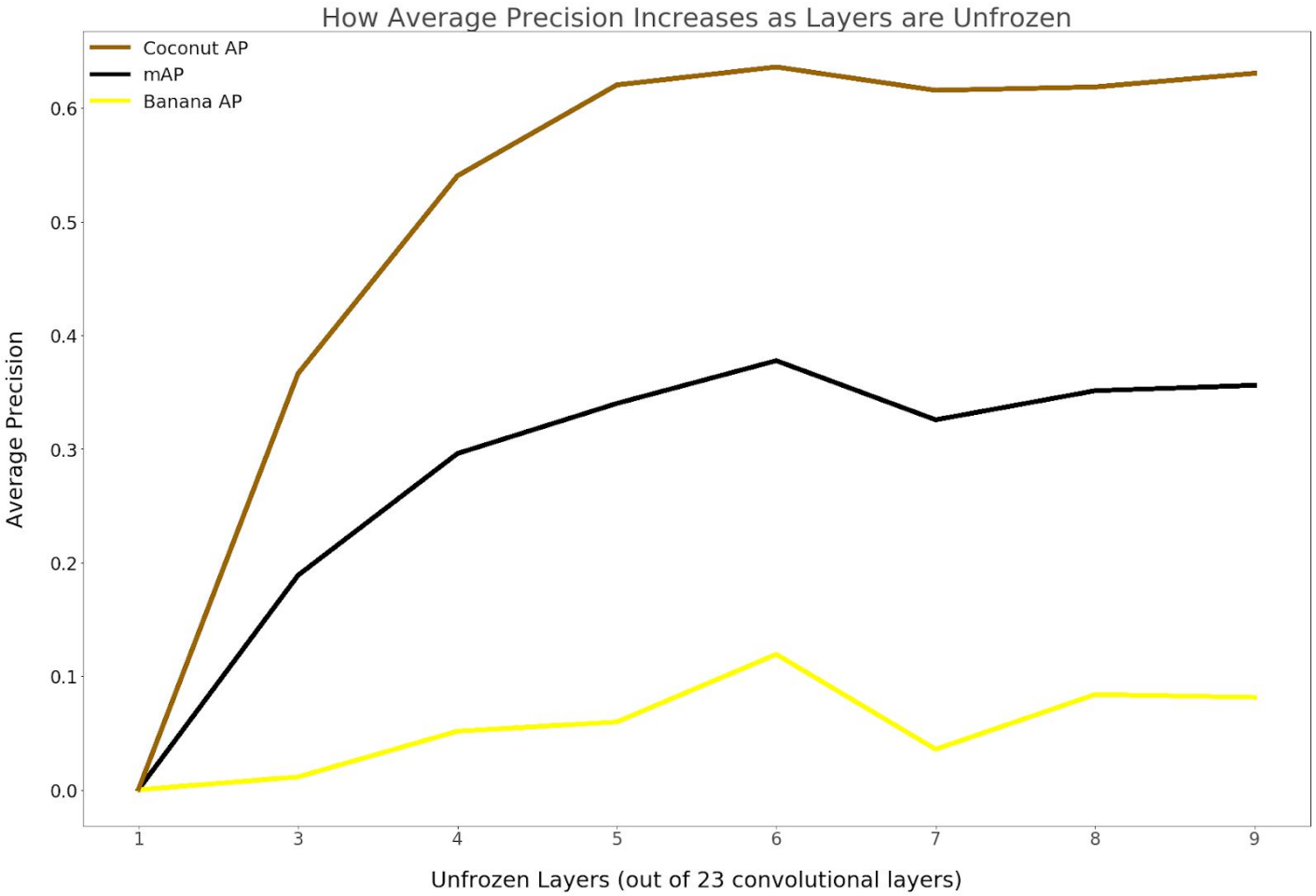

Figure 2: A plot of the results of the iterative unfreezing of layers process described in the Transfer Learning section, showing a slight peak at 6 unfrozen layers.

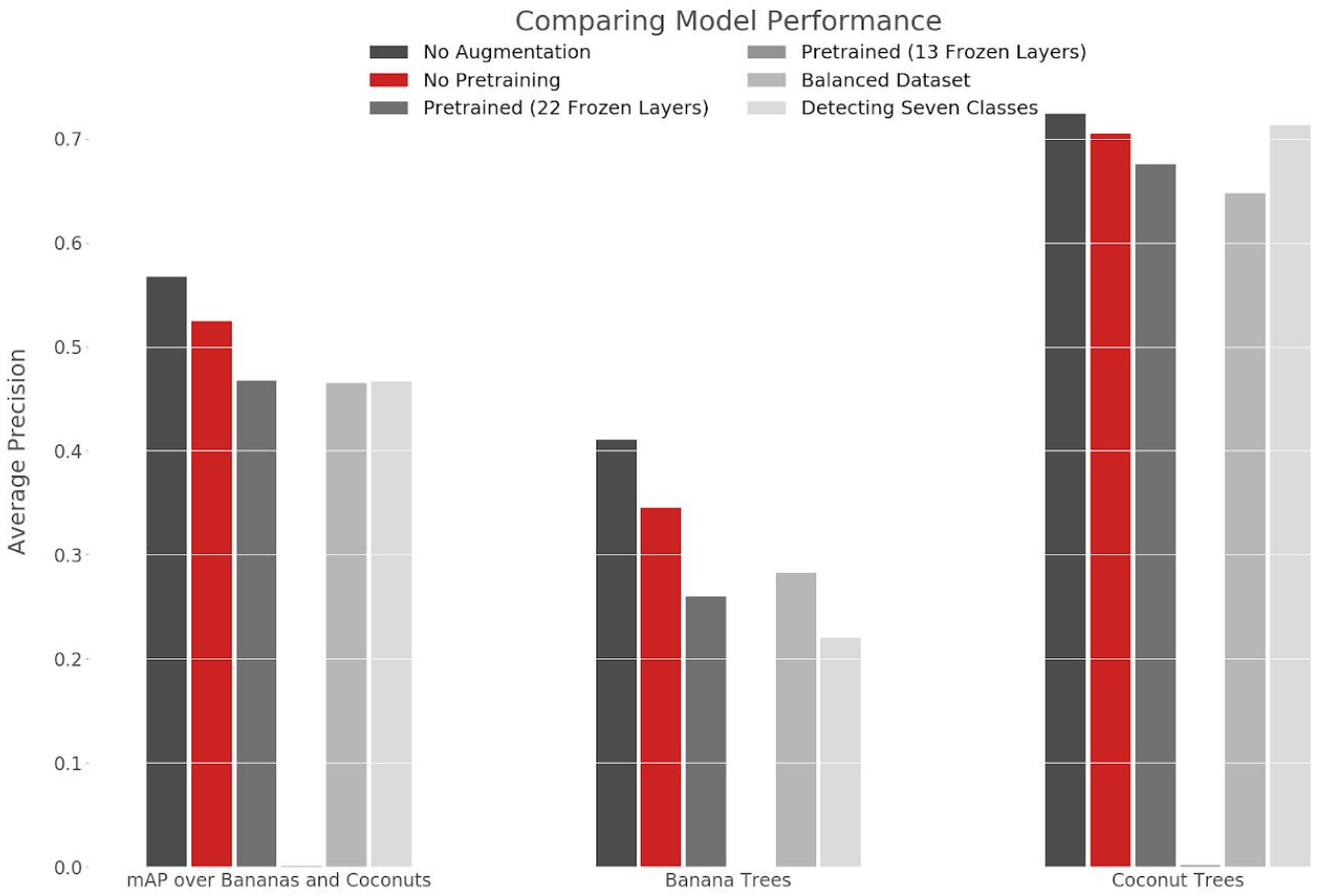

Figure 3: Comparison of mAP achieved by six of our models. The mAP of the model that was trained with a set of pretrained weights supplied and only the final classification layer unfrozen is barely visible as its results were abysmal.

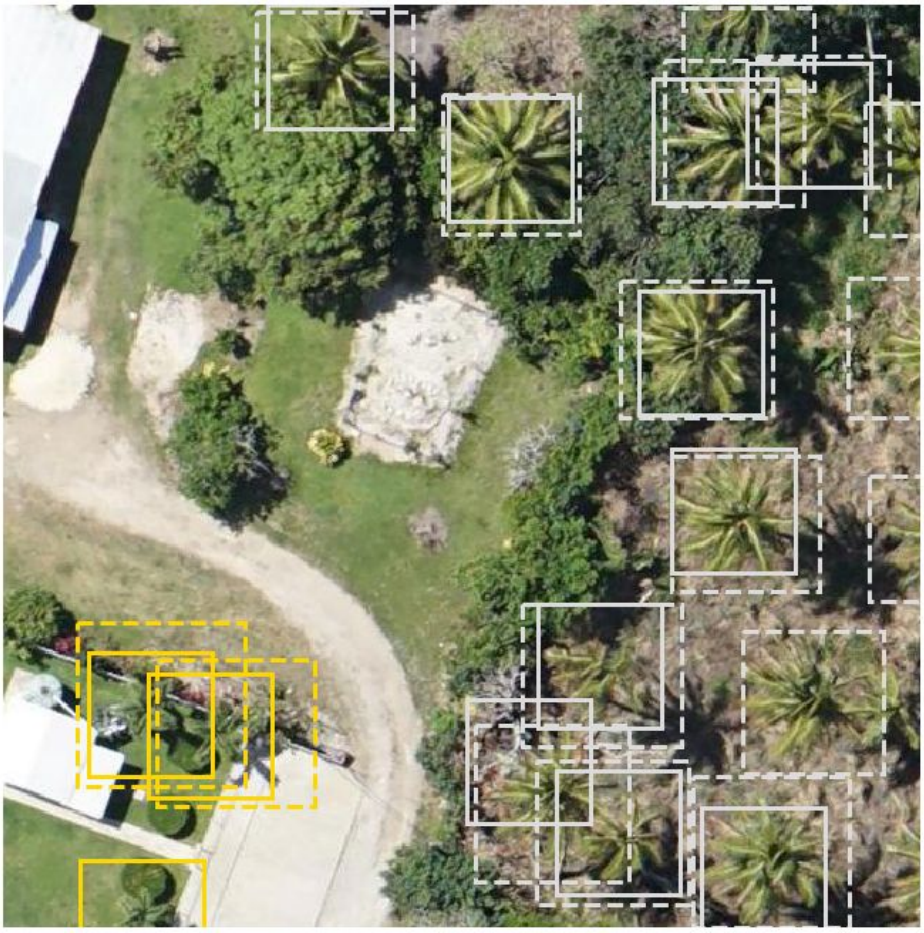

Figure 4: An example tile from our test show showing banana trees in yellow and coconut trees in white. Broken line boxes show detections and solid line boxes show ground truth.

Figure 5: A tile from our test set showing examples of detections our CNN made that did not appear in the ground truth labels. Broken line bounding boxes show our detections of coconut trees. Solid line bounding boxes show locations of coconut trees as provided in ground truth labels. We evaluated these detections as false positives, and so they negatively affected our average precision values.

Figure 6: Example of an aerial image of coconut trees unrelated to our original data set for which our model was unable to make any predictions.