# Object Detection with Mask R-CNN: Drone Images of Transmission Tower Components

**By: Daniel Foote, Filip Krutul, Ghadah Alshahrani, Zhiying Lu**

## Introduction

For this project, we chose to implement the Mask R-CNN object detection and segmentation model on the provided drone images of transmission tower components. At the beginning of the semester, students in the course annotated these images using VGG Image Annotator (VIA). These annotations were then compiled by the instructors, and fine-tuned to serve as a suitable dataset for object detection. To train an object detection model, we used Matterport's implementation of Mask R-CNN, which is written in Python 3, and runs on a TensorFlow and Keras backend. In order to prepare training and testing code, we followed a four part series of Youtube videos created by Data Lab at TXST (the link to the channel is provided in the references section). Additionally, we used Google Colab for training and inference due to GPU availability, which made the running time of training manageable. The sections below will briefly detail the process behind the implementation, as well as display concrete results obtained in the project.

## Data Preprocessing, Separating, and Parsing

In order to produce a manageable dataset, we manually isolated a subset of the drone dataset and split it into smaller images using the provided image splitter code (courtesy of Dr. Yan). First, the selected images and their corresponding annotations were isolated from the entire set by hand. During this process, we made sure to create a concise set of high quality images which contained as many distinct and visible instances of each class as possible. In order to split the images, we uploaded them to a personal GitHub account, along with the provided splitter code. This was done to simplify the directory structure within Google Colab, and store all the data in one place. The implementation details of running the provided splitter code can be found within the "splitter.ipynb" notebook file. Then, the images and their corresponding annotations were manually divided into training, testing, and validation sets with roughly a 0.7 - 0.15 - 0.15 split. The three sets were then placed in a Google Drive folder that can be mounted and accessed by Google Colab. In order to parse the data, we wrote a function which takes in the JSON annotation files, and parses them into a coco-like format that the model can understand. More details about the parsing function can be found in the training file.

## Training the Model

In order to train the model, we used information from the aforementioned series of videos and the Balloon.py functions: Load_Dataset and Load_Masks. Configuration was taken from the run-prediction-fromchopped.ipynb file provided by the instructor. The only addition to the Balloon.py functions was the implementation of multiple classes. Instead of returning a np.ones array, Load_Masks returns a Numpy array of the different class IDs. Next, we prepared both the training and validation set for training and then created a model object to train. The training process used 10 epochs with 2000 steps per epoch. This training session was only done on the heads layer, and the weights were saved to the model directory. However, a manual save option was implemented if needed. TensorFlow 1 was used for training, as recommended by the instructor. Specific implementation details can be found in the training file.

## Making Predictions Based on Trained Model

In order to test the performance of the model, we used information from the series of videos mentioned in the introduction. Implementation details can be found in the inference file. The Pillow, skimage.io, and listdir packages were used in addition to the standard Mask RCNN package. Configuration was kept the same except for Detection_Min_Confidence which was set to 0.8 to reduce incorrect masking. The same Load_Dataset and Load_Masks functions were used to load in the test data. Skimage.io was used to read in the original image and display for comparison against the Ground Truth and the Inferenced Images. To inference an image the Model.detect function was called and then visualize.display_instances was used to visualize the results. Lastly, we used mAP score to test the validity of the model. Specific implementation details can be found in the inference file

## Evaluating Performance

In order to quantitatively measure the performance of our model, we calculated the mAP (mean Average Precision) score. The mAP score compares the object detections of our model to the ground truth data by measuring the overlap between predicted and actual boundaries, and reporting the metric as a value between 0 to 1, corresponding to the proportion of images that were correctly detected. Implementation details can be found in the inference file. Using the testing code provided by the instructor, we also calculate the possibility of each class IDs.

## Limitations

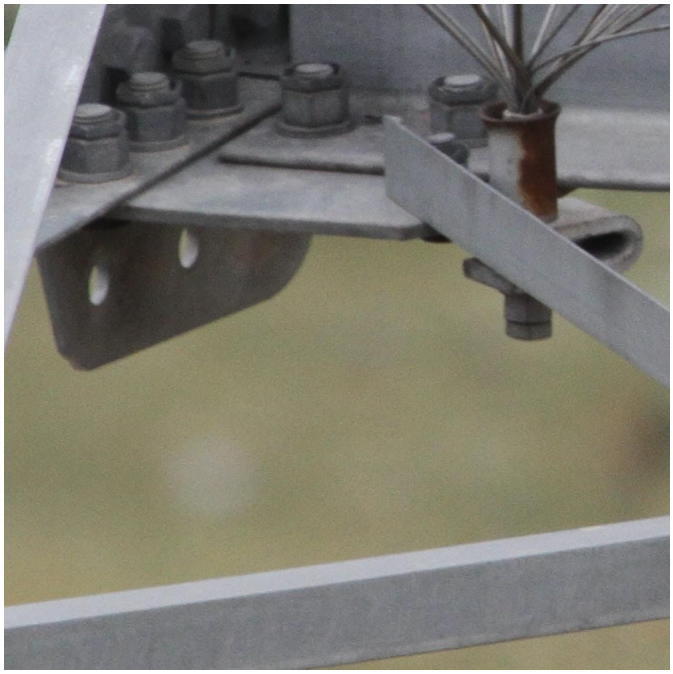
The primary limitation of our model was our inability to merge the split images back into their original form. Despite the fact that we were not able to accomplish this part of the task, we were able to obtain reasonably high quality object detection on the split images, which reflect the dataset as a whole. Additionally, we did not use the entire drone data set. Doing so could have improved the precision of the model and provided even better detection.

## Results & Conclusion

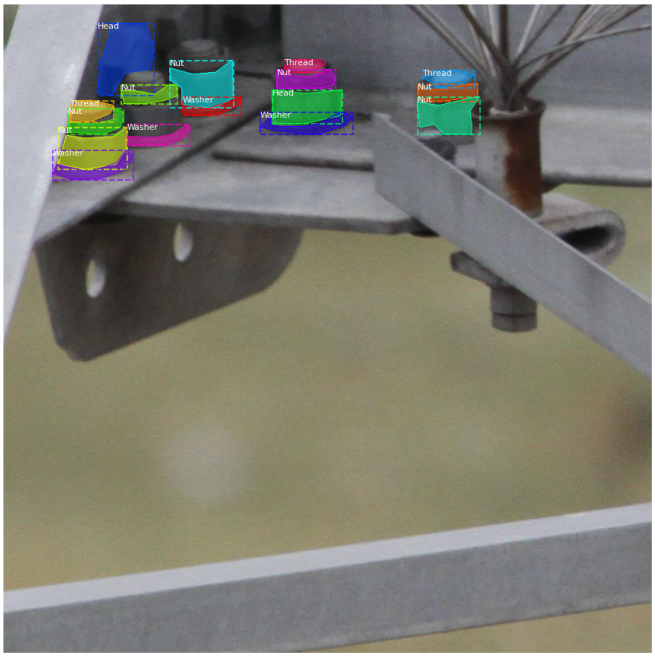
Below, you can find several images of object detections performed by the model, next to the original image and ground truth pictures for comparison (more sample predictions can be found in the ExtraImages folder attached to the submission:

**Image A:**

Original



Ground Truth:




Inference:

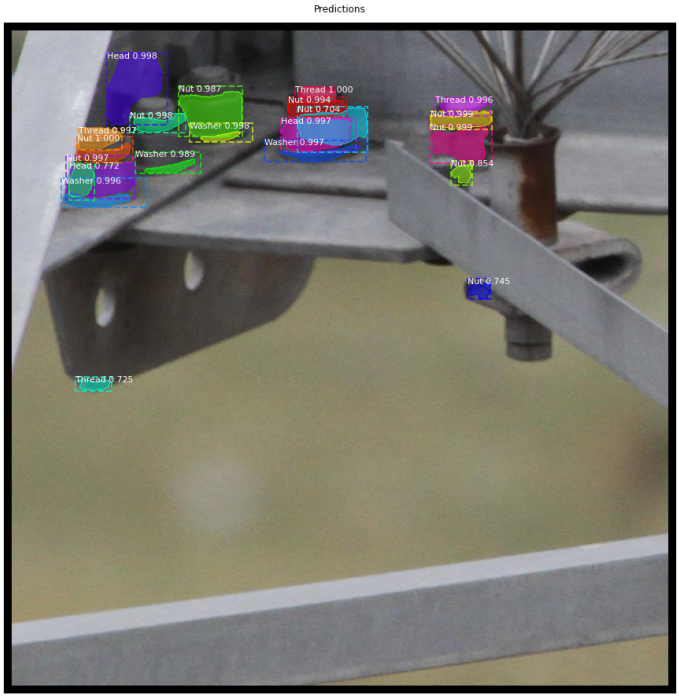
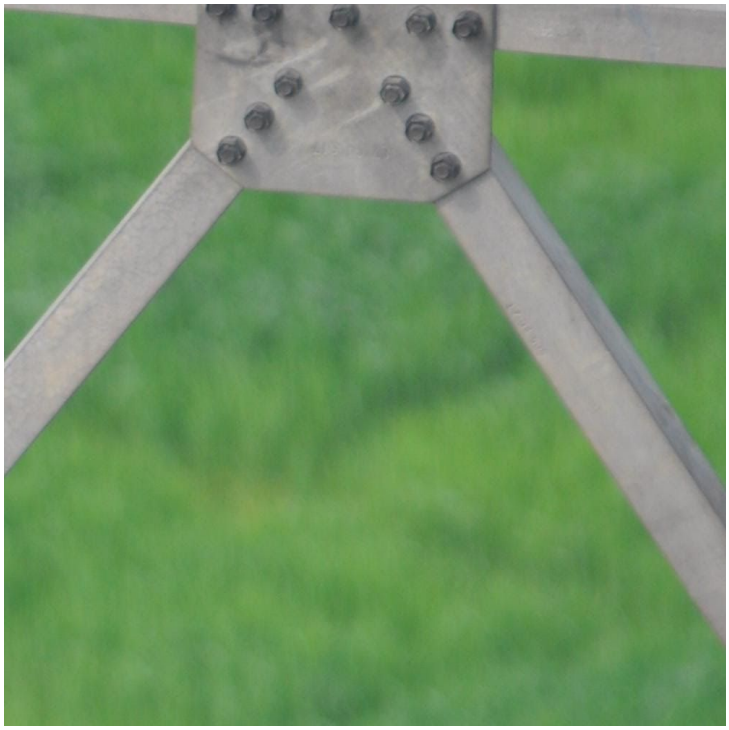
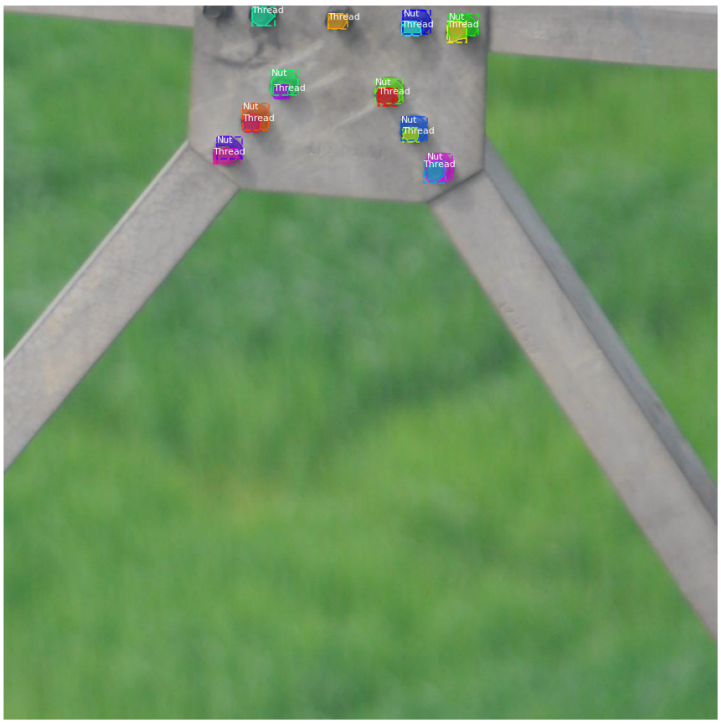


**Image B**

Original:



Ground Truth:



Inference:

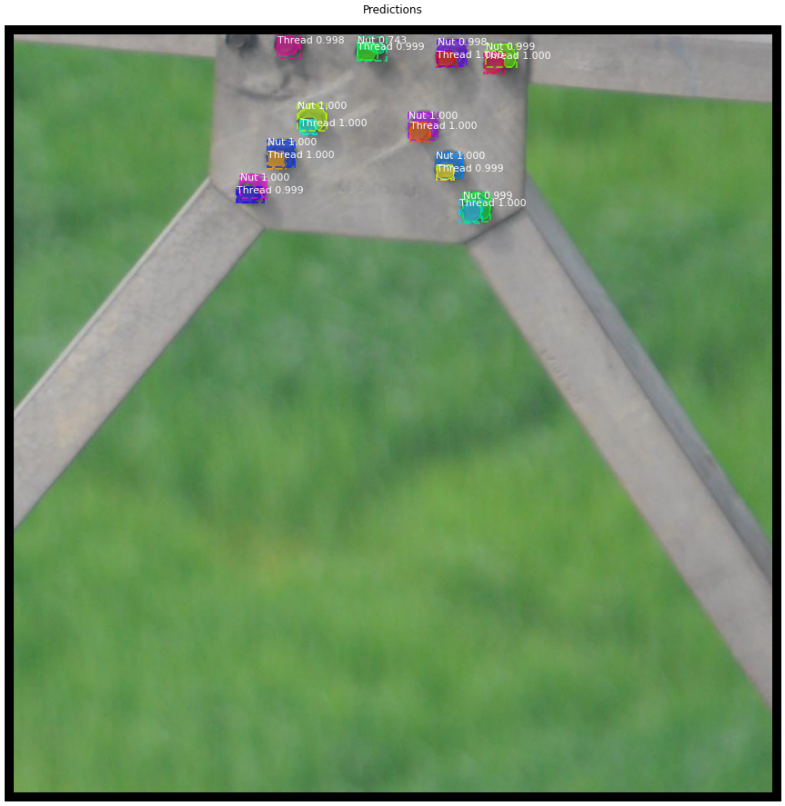

Predictions
Thread 0.998
Nut 0.743
Thread 0.999
Nut 0.998
Thread 1.000
Nut 0.999
Thread 1.000
Nut 1.000
Thread 1.000
Nut 1.000
Thread 1.000
Nut 1.000
Thread 1.000
Nut 1.000
Thread 0.999
Nut 1.000
Thread 0.999
Nut 0.999
Thread 1.000

**Image C:**
Original:

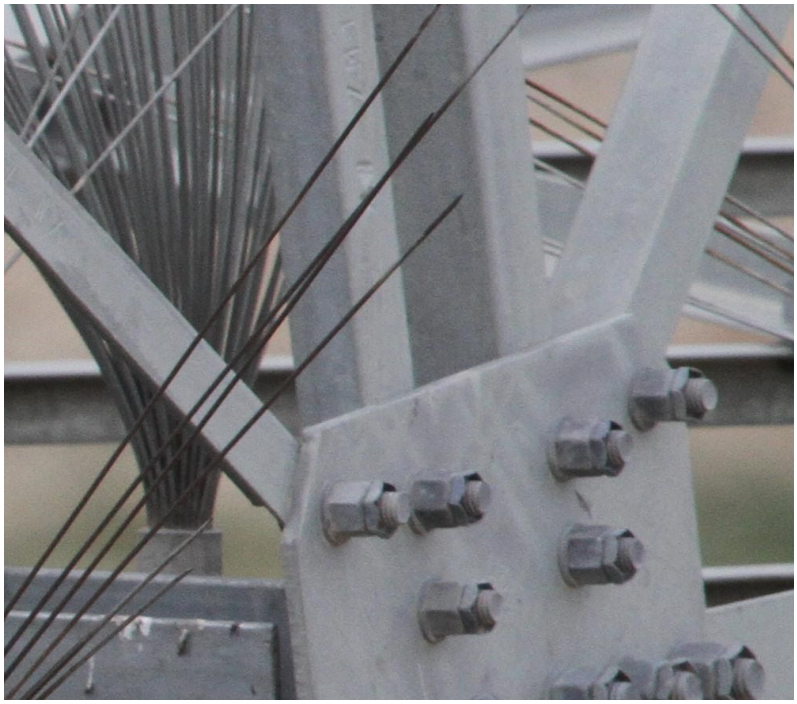
Ground Truth:

Inference:

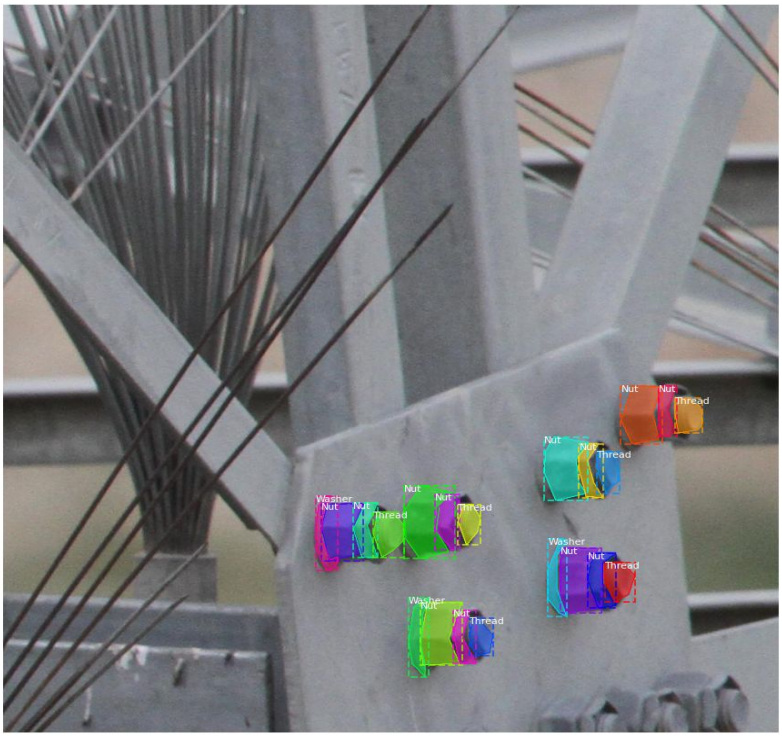

Nut 0.987
Thread 1.000
Thread 0.969
Head 0.999
Nut 0.992
Head 0.939
Nut 0.993
Thread 0.998

**Image D:**

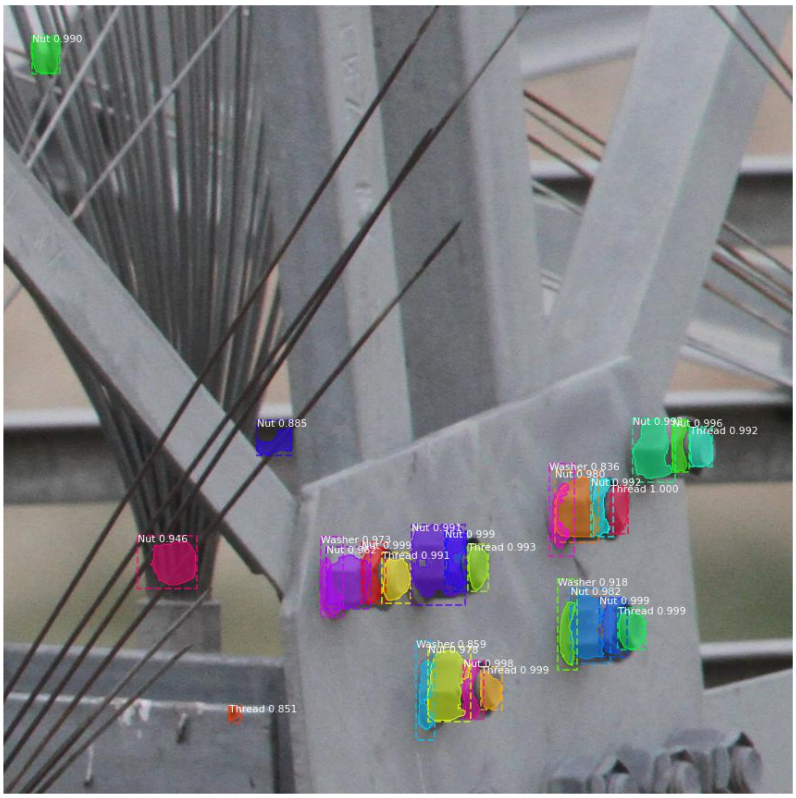
Original



Ground Truth



Inference




As seen in the sample images above, our model detected quite a few redundant objects that are not a part of the original annotation set. For example, in the image directly above several parts of the wiring are labeled as “nut”. In general, the model tended to over-detect objects that should not be detected. The detection of proper images, however, was quite accurate, with a mAP score of 0.972.

## References


1. https://www.youtube.com/channel/UCcrDBDkozBMilFnwLt2TpIA
2. Testing code from the instructor.
3. https://github.com/matterport/Mask_RCNN/blob/master/samples/balloon/balloon.py