Wadhwami AI Bollworm Counting Challenge

Sam Pastoriza, Haley Roberts, Scott Johnson, Matt Gnanadass, Matt Harding

December 2, 2022

Introduction

In 2017, a bollworm infestation in India resulted in major cotton farming losses. After this infestation, a company (Wadhwani AI) developed a mobile app in which farmers could upload pictures of bollworms caught in traps and receive advice on how much pesticides to use based on their level of infestation. While the company's app has been somewhat successful, they are seeking to improve its accuracy. To do this, they worked with Zindi, an Africa-based data science community and competition platform, to host the modeling problem as a competition. The "Wadhwani AI Bollworm Counting Challenge" prompts competitors to create a model to count the number of two types of bollworms (the American Bollworm, or ABW, and the Pink Bollworm, or PBW) found in images submitted by farmers.

Hence, this study seeks to build an efficient and accurate model to count the number of each type of bollworm in images. Successful counting of bollworms can greatly benefit the cotton farming community in India and reduce damage from infestations.

The data for the study is provided through Zindi's competition site and consists of three csv files (training dataset, testing dataset, bounding box information) and an image dataset. Each row of the training dataset contains an image ID and the number of each type of bollworm contained in the image. The testing data provides only the IDs of the images to be analyzed. The bounding box csv contains geometric information of bounding boxes around each bollworm in each image of the training dataset. Finally, the image dataset contains all of the training and testing images. The final results will be in the format of image IDs combined with the number of each type of bollworm found in each image.

Methods

Overview

Object detection is a common topic of computer vision and many algorithms have been developed to approach this goal. This study will consider three such algorithms: YOLO (You Only Look Once), Faster R-CNN, and Mask R-CNN implemented with Detectron2. The details of each algorithm's implementation will be described in their respective sections below. Each of these methods utilize convolutional neural network architecture to detect objects in images. The algorithms identify detected objects by generating labeled bounding boxes around each object, labeled for the type of object contained. After objects are detected, counting each type of object in an image is simple.

The process for each of the algorithms follows a standard machine learning approach. Each network is trained on the training images and their pre-labeled bounding boxes. Then, as discussed in each respective

section below, actions such as saving weights across epochs are used to prevent overfitting. Finally, an MAE loss function compares the counted number of bollworms to the actual number of bollworms to evaluate model performance.

YOLO (You Only Look Once)

YOLO is a popular method that approaches object detection in images in a unique way as compared with several other object detection algorithms. Rather than viewing each image at different scales and different regions, YOLO only looks at each region of images once (hence, the name of "You Only Look Once"). The method "appl[ies] a single neural network to the full image" by "divid[ing] the image into regions and predict[ing] bounding boxes and probabilities for each region," where the "bounding boxes are weighted by the predicted probabilities" (Redmon & Farhadi, 2018). Because of this single-network approach, YOLO can perform faster than many other detection algorithms.

For this project, YOLOv5 from Ultralytics was utilized, as accessed through their official Github repository. First, the bounding boxes for the detected objects in the images are determined and the x- and y-coordinates are normalized using the width and height of each image. Then, by using a transfer learning approach, the model is trained to recognize each bollworm over 10 epochs on 9737 images, with a 90-10 split for training and validation data. With the model trained, training versus validation loss are visualized to determine the number of epochs at which the model started to overfit. Using saved model weights at the specified epoch, the model was run on the competition's testing data and the results retrieved. These results are discussed below.

Faster R-CNN

Faster R-CNN is an object detection algorithm extending from the original R-CNN and Fast R-CNN algorithms. R-CNN is a region-based convolutional neural network that considers many regions across an image and classifies each region as being an object of interest or not. Due to the extensive nature of this regional selection approach, R-CNN has several drawbacks. A Facebook AI researcher developed Fast R-CNN to counteract several of these drawbacks. Fast R-CNN added an ROI pooling layer, allowing each image to be processed in a single stage rather than the multi-stages, thus increasing the speed of the algorithm (Gad, 2020). Finally, Faster R-CNN extended from Fast R-CNN by including "a fully convolutional network that generates proposals with various scales and aspect ratios", called the region proposal network, or RPN (Gad, 2020). This region proposal network "implements the terminology of neural network with attention to tell the object detection (Fast R-CNN) where to look" (Gad, 2020). Several other updates to the Fast R-CNN method allow Faster R-CNN to perform faster and more efficiently.

This project utilized the PyTorch implementation of Faster R-CNN which is based on the Faster R-CNN algorithm discussed in the article "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks" (Ren et al., 2015). As mentioned, this algorithm uses "a Region Proposal Network (RPN) that shares full-image convolutional features with the detection network, thus enabling nearly cost-free region proposals" (Ren et al., 2015). The RPN is used in the first stage of the algorithm, followed by a utilization of a Fast R-CNN detector which outputs the bounding boxes and class labels of a

proposed region (Megne Choudja, 2022). The details behind this method are shared in the mentioned article.

Here, the Faster R-CNN algorithm was implemented using PyTorch with a ResNet50 architecture backing the model. First, the bounding boxes for the detected objects were determined and the x- and y-coordinates normalized. Next, a transfer learning approach was utilized where weights that were pre-trained on the COCO dataset (a large dataset containing "images of complex everyday scenes containing common objects in their natural context") were used in the model, but retrained to predict one of the two classes of bollworms (Lin et al., 2014). Then, the model was trained with 10 epochs on 9737 images, with a 90-10 split for training and validation data. After every 2 epochs, the model's weights were saved to prevent overfitting when running the model on the test data. After the model was trained, the training versus validation loss was visualized to determine the number of epochs at which the model started to overfit. Using those saved model weights at the specified epoch, the model was run on the competition's testing data and the results retrieved. These results are discussed below.

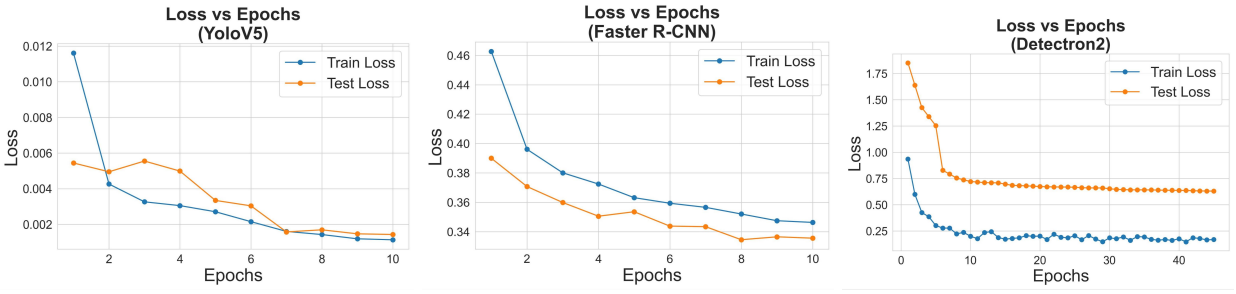
Mask R-CNN (implemented with Detectron2)

The final algorithm used is Mask R-CNN which is "a conceptually simple, flexible, and general framework for object instance segmentation" extending from Faster R-CNN (He et al., 2017). Mask R-CNN diverges from Faster R-CNN in that it outputs class labels, bounding boxes, and an object mask which "require[s] extraction of much finer spatial layout of an object" (Megne Choudja, 2022). The algorithm "is simple to train and adds only a small overhead to Faster R-CNN" (He et al., 2017).
For this project, Mask R-CNN is implemented through the Pytorch-based Detectron2 framework built by Facebook AI researchers. To utilize this model, the data was first formatted appropriately by determining all of the masks and associated bounding boxes for each object detected in the image. The training dataset provided the masks so these were utilized and the bounding boxes created by finding the largest and smallest pairs of x and y values. Unlike the previous two approaches, the bounding boxes and masks were not normalized per specifications. However, similar to the previous two methods, a transfer learning approach was taken to train the model to detect the two types of bollworms. The model was trained on a recommended 40 epochs and the training and validation loss were tracked per epoch. Additionally, the model's weights were saved every 5 epochs. After the model was trained, the training versus validation loss was visualized to determine the number of epochs at which the model started to overfit. Finally, using those saved model weights at the specified epoch, the model was run on the competition's testing data and the results were retrieved. These results are discussed below.

Results

Loss Over Epochs

All three models were trained on the training dataset and used to predict the number of each type of bollworm in each of the testing images. Each model was trained over several epochs to optimize the results. Figure 1 shows the loss over epochs for each model.



(a) (b) (c)

Figure 1: The loss at each epoch for (a) YoloV5, (b) Faster R-CNN, (c) Mask R-CNN implemented with Detectron2.

Here the loss over epochs for each of the three models is seen. The YoloV5 method results in a very small loss and does not appear to overfit the data, as the training and testing loss at the final epoch is approximately the same. The loss appears to taper off after 8 epochs, so only 10 epochs are run and shown here. The Faster R-CNN model continually improves across epochs but also appears to taper off in improvement after 8 epochs. The training loss is slightly higher than the testing loss, which may suggest slight underfitting; however, the difference is very small. Finally, the Mask R-CNN model sustains a somewhat large loss that only decreases a little after the 8th epoch. The large difference between the testing and training losses suggests overfitting.

Output of Models (Bounding Boxes and Predictions)

Each of the three models used produces bounding boxes around each object detected, along with a label of the object's classification (abw or pbw). The Mask R-CNN (Detectron2) and YOLO also provide a probability of prediction for each object. Figures 2, 3, and 4 show the resulting bounding boxes and predictions for a portion of one of the testing images as produced by the YOLO algorithm, Faster R-CNN algorithm, and Mask R-CNN algorithm, respectively. The entire image with its results for each of the three algorithms is shared in Appendix Figures 1, 2, and 3, but a cropped and zoomed-in version is shown here for readability of the predicted classifications.

Figure 2: Bounding box and bollworm classification of a portion of a test image as found by the YOLO algorithm (complete image shown in Appendix Figure 1).

Figure 3: Bounding box and bollworm classification of a portion of a test image as found by the Faster R-CNN algorithm (complete image shown in Appendix Figure 2).

Figure 4: Bounding box and bollworm classification of a portion of a test image as found by the Mask R-CNN algorithm, implemented with Detectron 2 (complete image shown in Appendix Figure 3).

The above figures show that the three models predicted almost the same results of bounding boxes and labels for this particular test image. Figure 4 shows that the Mask R-CNN algorithm incorrectly predicted an extra bollworm (in the top left corner, seen in the yellow box) that was not predicted by either of the other two models. However, overall it is encouraging to see the accuracy of the bounding boxes and classifications produced by the three models.

Model Performance (Mean Absolute Error)

The final tuned models for each method (YOLO, Faster R-CNN, and Mask R-CNN) were used to predict the number of each type of bollworm for each test image. To follow the Zindi competition's rules, the metric to evaluate the success of each model is a modified Mean Absolute Error (MAE). Since each image will have a count of ABWs and PBWs, each image's absolute error is found by summing the total error across the two classes. Then, the overall MAE across all images is found and the overall metric is calculated with Equation 1.

$$\frac{1}{|I|}\sum_{i\in I}\sum_{p\in P}|\hat{p}_i|_p - y_i|p|$$

Equation 1: The modified Mean Absolute Error (MAE) metric for model results (per Wadhwani AI Bollworm Counting challenge).

The variables in this equation are explained in Table 1, as described by the Zindi competition's website.

Table 1: Description of variables used in modified Mean Absolute Error formula (Wadhwani AI Bollworm Counting challenge).

Variable	Description
I	The set of images
P	The set of pest labels
y	The ground truth count
$\hat{y}$	The predicted count
subscript vertical bar notation	Read as the prediction at image i, restricted to label p

The final MAE metric for each of the methods' final tuned models were calculated and the results are shown in Table 2.

Table 2: Mean Absolute Errors of final tuned model for each algorithm used.

Model	Mean Absolute Error
YOLO	2.380
Mask R-CNN (Detectron2)	2.605
Faster R-CNN	2.997

The results shown in Table 2 indicate that the best model performance occurred with the YOLO algorithm, with a mean absolute error of 2.380. This error placed the model as the 57th best-ranked submission to the Zindi competition. A score of 2.380 indicates that, on average, the model's calculation of the number of bollworms was incorrect by 2.380 bollworms. This is not a large amount and suggests that the model performs well and could be used to indicate the number of bollworms in images accurately. However, more discussion with bollworm-infestation experts would be needed to quantify what margin of error in counts are acceptable for the results to be usable in prescribing pesticide usage given the number of bollworms seen. For example, if the amount of pesticides changes based on every single incremental number of bollworms found in a bollworm trap, then this model is not very useful as it may produce an inaccurate number of bollworms that results in the wrong amount of pesticides being dispensed. However, if the threshold for changing pesticides amounts is every 10 bollworms, then perhaps this model is accurate enough for effective use.

Conclusions

Overall, this study aimed to build an efficient and accurate model to count the number of two different types of bollworms in images. However, rather than merely achieve this goal by creating one single model, this project created and employed a total of three different models to pursue this goal. This explicit choice to implement a range of various object detection models provided strengthened justification for the final model recommended for use and enabled a deeper understanding of the problem at hand. The training and testing of the three separate models allowed for easy and observable comparisons between the mean absolute error achieved by each model, as displayed in Table 2. Thus, a final recommendation of the YOLO model is easily justified, as it resulted in the lowest mean absolute error evaluation metric. Additionally, the final bounding box results of each model, as displayed in Figures 2-4, provided insights into the possible shortcomings in the models. Specifically, Figure 4 displays that the Mask R-CNN algorithm incorrectly predicted an extra bollworm that was not present in either of the other two models' predictions. This exemplifies two of the challenges that each model faced in making predictions on images with overlapping and randomly-oriented bollworms. While this specific image did not have overlapping bollworms, other images did which is why the model may have made such a prediction. Other challenges faced by the models were the low resolution of the images and occasional missing limbs on the bollworms (various detached bollworm limbs can be observed on the piece of paper in Figures 2-4). In summary, the successful implementation of these three separate models justified final recommendations and provided insights into the major challenges of this problem.

Moving forward, it would be highly recommended that some of the challenges that likely hinder the overall accuracy of each model be addressed in future training and testing datasets. Specifically, the easiest challenge to fix would be to avoid any overlap of bollworms in the images. If the farmers who upload images of bollworms properly separate bollworms on the piece of paper, the models can make more accurate predictions. The challenge of low-resolution images also could potentially be addressed by simply increasing the resolution of each image. However, it is recognized that not all of the farmers taking these images have the resources and technology for such higher-quality images. Moreover, the final challenge of missing limbs is one that is also difficult to fully fix, as the bollworms may be losing limbs by attempting to escape the traps. Thus, it is not entirely possible for farmers and those taking these images to address this problem. Additionally, alongside these physical steps that might benefit the models, there are a few improvements to the models that might better enable lower mean absolute error values. The first and most obvious would be a greater extent of hyperparameter tuning on each of the models. Thus far, each model has been created and trained using its default parameters. Although these default parameters are successful in general object detection, some level of additional hyperparameter tuning may benefit the specific bollworm detection within this study.

Finally, although this study's end goal was to aid the cotton farming community in India, the methods in utilizing an app to collect images for the training of object detection models could have major benefits for other large-scale problems both within agriculture and other industries. In reality, this study is a perfect example of the benefit that convolutional neural networks, object detection, and other associated machine and deep learning techniques can provide to the general public when there is data readily available.

References

Gad, A. F. (2020). Faster R-CNN Explained for Object Detection Tasks [web log]. Retrieved November 28, 2022, from https://blog.paperspace.com/faster-r-cnn-explained-object-detection/.

He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2017). Mask R-CNN. *2017 IEEE International Conference on Computer Vision (ICCV)*, 2980–2988. https://doi.org/10.1109/ICCV.2017.322

Lin, T.-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C. L., & Dollár, P. (2014). Microsoft COCO: Common Objects in Context. *ArXiv*. https://doi.org/10.48550/ARXIV.1405.0312

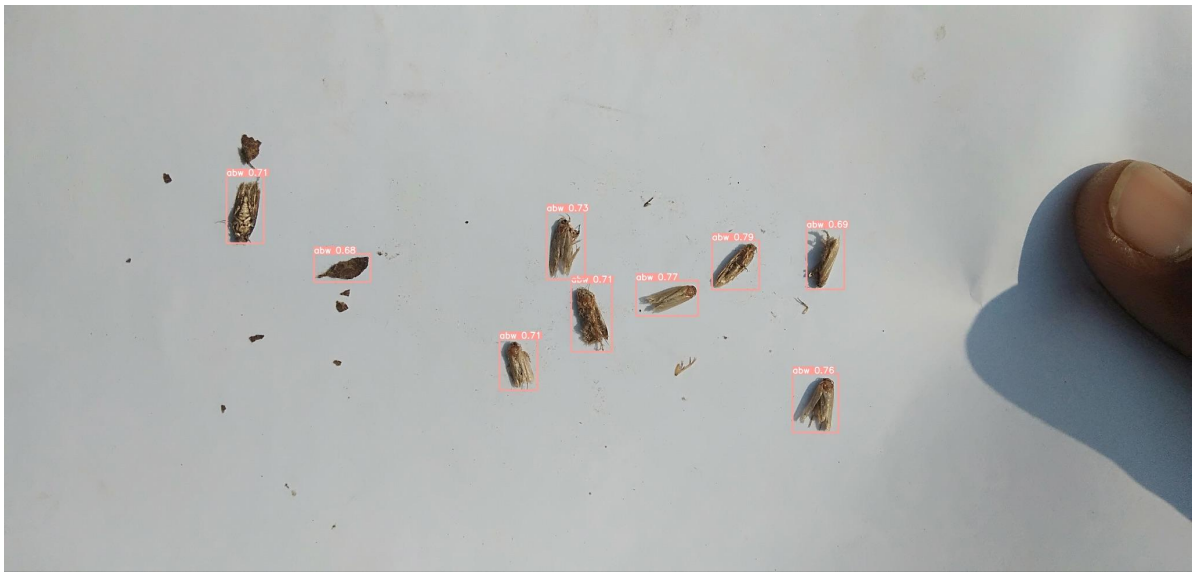
Megne Choudja, P. O. (2022, May 22). Mask-RCNN for Object Detection with Detectron2 [web log]. Retrieved November 28, 2022, from https://medium.com/mlearning-ai/mask-rcnn-for-object-detection-with-dectectron2-46f62f6b7826.

Redmon, J., & Farhadi, A. (2018). YOLOv3: An Incremental Improvement. *ArXiv*. Retrieved from https://pjreddie.com/darknet/yolo/.

Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. https://doi.org/10.48550/ARXIV.1506.01497

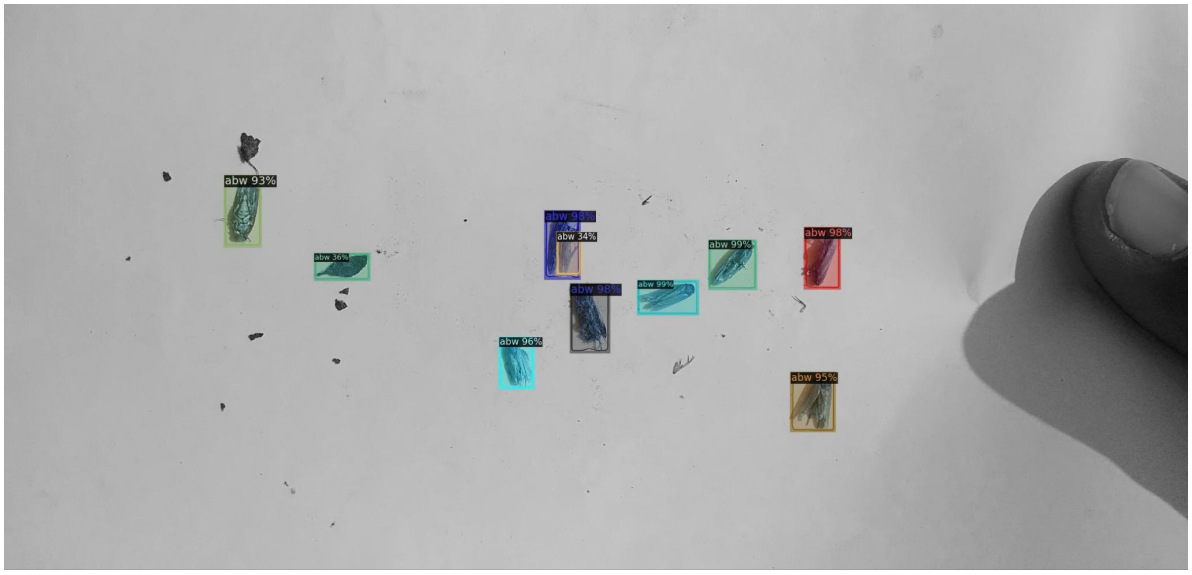
Wadhwani AI Bollworm Counting challenge. Zindi. (n.d.). Retrieved October 5, 2022, from https://zindi.africa/competitions/wadhwani-ai-bollworm-counting-challenge

Appendix



Appendix Figure 1: Bounding box and bollworm classification of a test image as found by the YOLO algorithm.

Appendix Figure 2: Bounding box and bollworm classification of a test image as found by the Faster R-CNN algorithm.



Appendix Figure 3: Bounding box and bollworm classification of a test image as found by the Mask R-CNN algorithm, implemented with Detectron 2.