Fruit Detection and Yield Estimation Based on Yolo v2

Team Member:

Zhiyu Wen (zw1305) Tianyou Zhou (tz895)

Github Repo:

https://github.com/ConnorWen9398/Fruit_Detection.git

Weights:

https://drive.google.com/open?id=1Tpl_IscaV1Ybd6pc3vJQ38mDZVMszD_o

1. Introduction

In the machine learning research community, applications in robotics have always been a hot topic, and object detection is one of them. For many automated robotic systems, an efficient and reliable imaged based object-detection subsystem is acting as a guide for decision making. Researchers from the Australian Centre for Field Robotics, The University of Sydney, released the ACFR Orchard Fruit Dataset [1] which contains images collected by an unmanned ground vehicle with annotations for three kinds of fruits. The objective of this project is to identify individual apples in the images and estimate the total number of apples over the dataset by accumulating the counts. The development environment is based on Ubuntu 16.04 with Cuda 9.1, and a desktop with GTX 1080 Ti and i9-7900X is used for training and testing supported by NYU Robotic Design Team.

2. Related Work

In [2]-[3], the method used for fruit detection is Faster R-CNN, which merges region proposals and object classification and localization into one unified deep object detection network. In [4], the authors demonstrated that using a Conditional Random Field to model color and visual texture features from multi-spectral images for sweet pepper segmentation. However, the relative low training and processing speed of Faster R-CNN and CRF may limited the operability of the detection system. A new approach Yolo v2 is proposed recently which framed object detection as a regression problem to spatially separated bounding boxes and associated class probabilities [5]. Using a single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation, Yolo is able to processes high-resolution images at 40-90 FPS on Titan X and has a mAP of 78.6% on VOC 2007 and 48.1% on COCO test-dev.

3. Yolo v2

Unlikely other detection systems, Yolo has only one single convolutional network simultaneously predicting multiple bounding boxes and class probabilities for those boxes, which means it is end-to-end network. Yolo trains on full images and directly optimizes detection performance, which makes an object detection problem be a regression problem. Basically, Yolo's neural network could be able to be seen as two parts. One is the initial convolutional layers of the network which extract features from the images. The other is the fully connected layers which

predict the output probabilities and coordinates. (In Yolo v2, it was instead of by Anchor Boxes just like Faster R-CNN used to avoid missing spatial information caused by fully connected). The network architecture is inspired by the GoogLeNet model for image classification including 24 convolutional layers followed by 2 fully connected layers (replaced by Anchor Boxes in Yolo v2). The full network is shown in Figure 1.

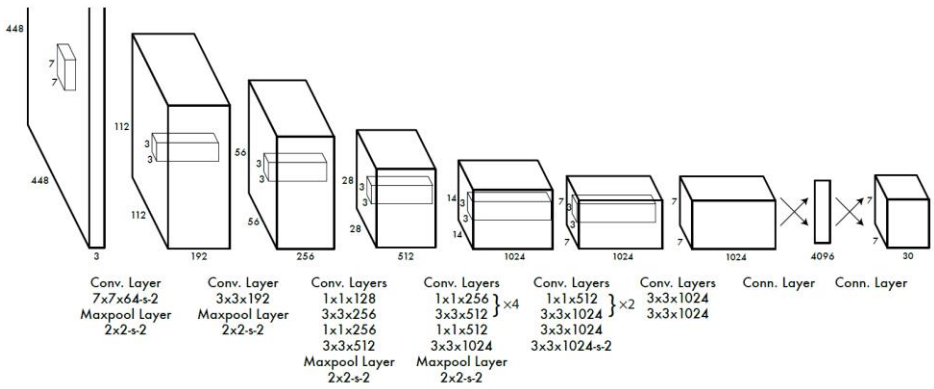

Figure 1. The Fully Network of Yolo

4. Detection Using Yolo v2 Pre-trained Model

The parameter of the Yolo v2 pre-trained model is storage in *yolo.weights*, which is trained for identify 80 classes including apple. Since the images in the dataset are in relative low quality and not in a normal style of the apples in everyday life, the Yolo v2 pre-trained model does not achieve an accepted result, and the detection results also contain irrelevant classes including bicycle, motorbike and etc. Figure 2 shows an example.

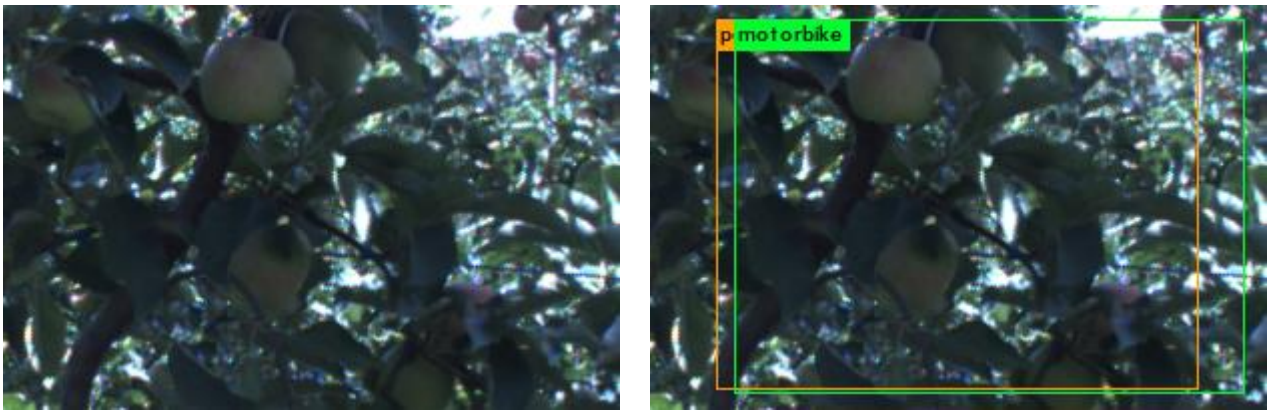

Figure 2. Detection Result of Pre-trained Yolo V2

By running

*./darknet detect cfg/yolo.cfg yolo.weights data/***.png -thresh 0.05*

*(Substitute ***.png with the filename to be processed)*

A labeled image named "*prediction.png*" containing all possible 80 classes will be generated storage in the root directory of the package, and it will be replaced when detecting another image.

By running

./darknet detect cfg/yolo.cfg yolo.weights -thresh 0.05

Processing multiple images with loading the neural networks once is supported, but the filename of next image is required to be entered manually after the previous detection completed.

5. Detection Using Modified Yolo v2 Model

To make the original model compatible with the dataset, following modification is made:

1) The output class is restricted to "apple", and the labeled images are storage in a folder named "*prediction*" with the original filename.
2) For multiple images, a txt file containing a list of filename can be read and processed automatically, and the total number of detected objects is returned.

With a threshold of 0.05, 225 of 554 apples are detected in the test set, but the results also contain a certain number of errors. All the results can be found in /saved_results/yolo_0.05, and Figure 3 shows an error sample.

Figure 3. Detection Result of Modified Yolo V2

6. Detection Using Retrained Yolo v2 Model

To improve the accuracy, we try to training Yolo v2 on the provided dataset following the below steps.

6.1 Data Annotation

ACFR Orchard Fruit Dataset contains 1120 images of apples, and we use 896 for training, 112 for validation, and 112 for testing. The list for each set can be found in *data/apples/sets*. The annotation for images in the dataset is in CSV format (example shows in Figure 4), while Yolo requires a .txt file (example shows in Figure 5) for each image, with each line representing a ground truth object in the image. Although there is transformation demo on the Yolo's website, it is from .xml to .txt for VOC dataset. Therefore, the first step for training is to transform the datasets to a correct format.

#item	c-x	c-y	radius	label
0	146.25	42.81	17.51	1
1	171.56	25.31	15.33	1

[object center in X] [object center in Y] [radius] [category number]

Figure 4. The original data format

```
0 0.474837662338 0.211930693069 0.113701298701 0.173366336634
0 0.557012987013 0.125297029703 0.0995454545455 0.151782178218
```

[category number] [object center in X] [object center in Y] [object width in X] [object width in Y]

Figure 5. Yolo v2 data format

Each .csv file in original datasets has the information including the coordinates of the center of the circle for each object, radius and labels. Since Yolo uses rectangle bounding boxes, the value of the width and the height of the boxes are set as circle's radius in this case. Meanwhile, Yolo uses the relative location of the object in the image for processing images with different size.

The python program for data format transformation can be found at */python*

csv2txt.ipynb: transforms original data format in csv. file to Yolo v2 data format in txt. file.

6.2 Configuration Files

cfg/apple.data: includes number of class, train and validation data path, and etc.

data/apple.names: includes the name of class.

cfg/yolo-apple2.cfg: describes the structure of the neural network. This file is modified based on *yolo-voc.cfg*, which is used to training Yolo on VOC dataset.

Line 3: set batch=64, this means we will be using 64 images for every training step

Line 4: set subdivisions=8, the batch will be divided by 8 to decrease GPU VRAM requirements.

Line 244: set classes=1, the number of categories we want to detect

Line 237: set filters = (classes + 5)*5, in our case filters=30

6.3 Iteration

A set of convolutional weights that are pre-trained on Imagenet is required for training YOLO v2. The file named *darknet19_448.conv.23* can be found in the Google drive link.

Start iteration by running:

./darknet detector train cfg/apple.data cfg/yolo-apple2.cfg darknet19_448.conv.23

The IoU and loss are reported after every training iteration as the format below:

*Region Avg IOU: **0.698363**, Class: 0.893232, Obj: 0.700808, No Obj: 0.004567, Avg Recall: 1.000000, count: 8*
***9002**: **0.211667**, **0.60730 avg**, 0.001000 rate, 3.868000 seconds, 576128 images Loaded: 0.000000 seconds*

- **0.698363** - IoU
- **9002** - iteration number

- **0.211667** - loss
- **0.60730 avg** - average loss

File *yolo-apple2_xxx.weights* will be saved to the \backup\ for each 100 iterations until 1000 iterations has been reached, and after for each 1000 iterations. The weights file can be found in Google drive link at the beginning of the report.

6.4 Evaluation

We train the model for 80200 iterations, and to evaluate the performance of the training model, we record the reported loss and average loss per iteration in file named "*avgerror.txt*" in the root directory, the value represent to iteration number is shown in Figure 6.

The python program for this part can be found at */python*

plotloss.ipynb: Plot the loss and average loss per iteration base on avgerror.txt

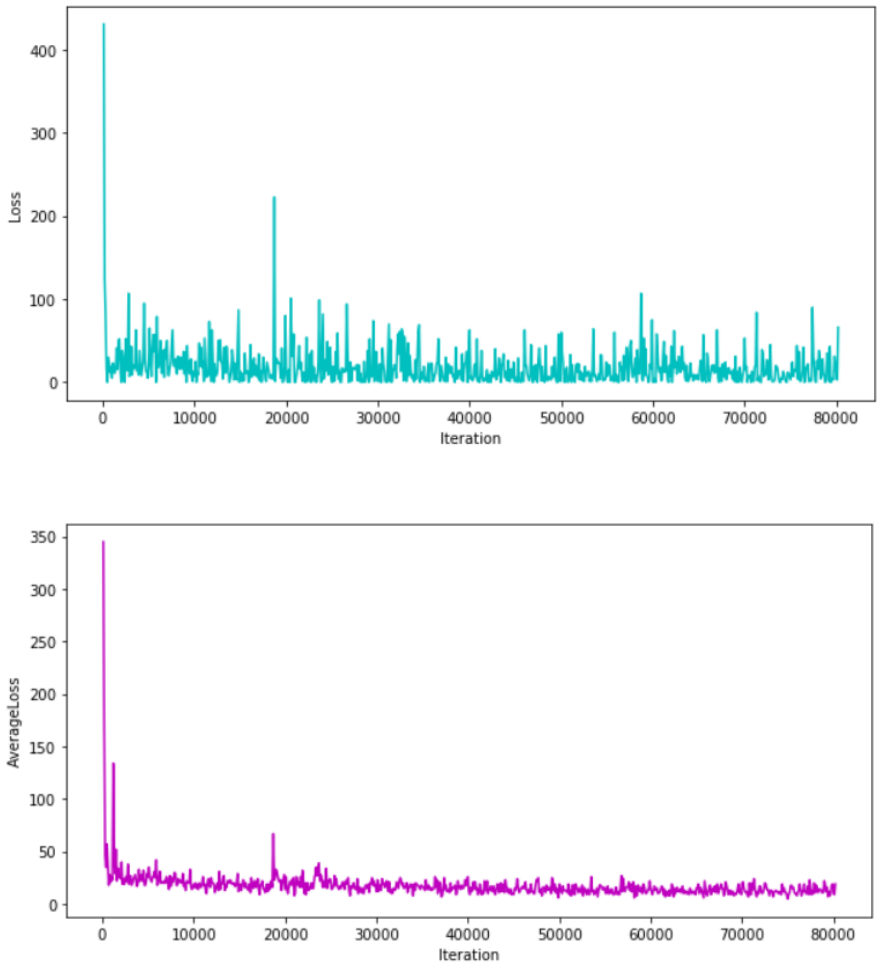

Figure 6. Loss and Average Loss per Iteration

6.5 Experiment

Yolo is supported for both CPU and GPU modes. The images in dataset are shaped in 308X202 pixels, and the time consumed for loading weights and processing single image is listed

in Table 1.

Table 1. Time Summary

	CPU	GPU
Loading Weights	4.57s	0.09s
Processing Single Image	3.97s	0.01s

Using the final weights file obtained from last step, 356 out of 554 apples, accounts for 64.3%, are detected in the test set with threshold of 0.03. The detection results are storage in */saved_results/final_0.03*. Changing the threshold as 0.02, 496 out of 554 apples, accounts for 89.5% are detected. Results are storage in */saved_results/final_0.02*. One sample with different threshold is shown in Figure 7&8.

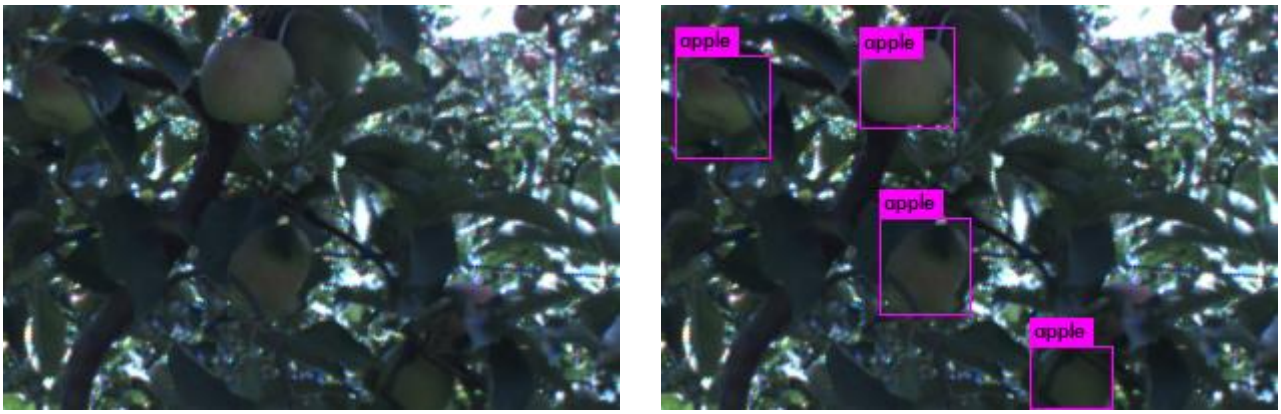

Figure 7. Detection Result of Retrained Yolo V2 with threshold = 0.03

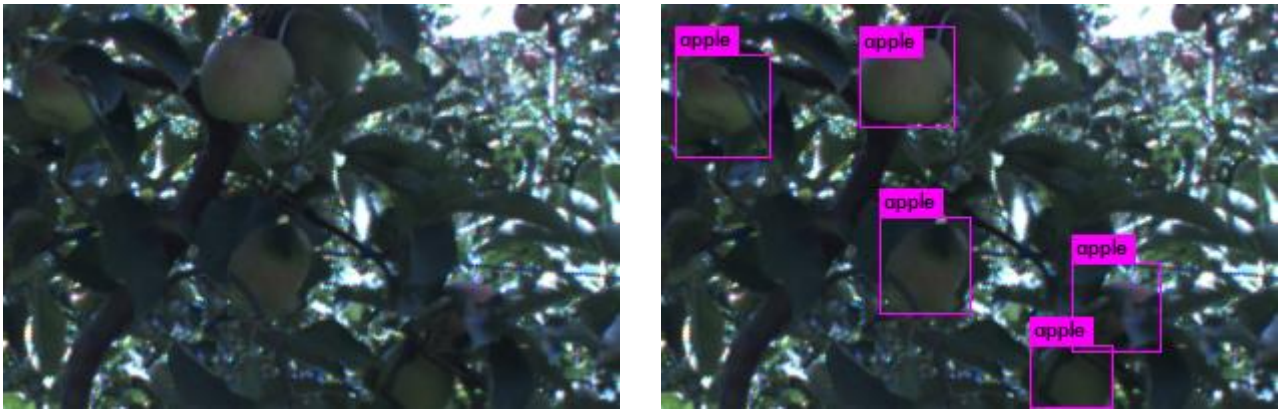

Figure 8. Detection Result of Retrained Yolo V2 with threshold = 0.02

7. Conclusion

A detection and yield estimation system for apple is developed in this project based on Yolo v2 model, and the processing speed is enough for real-time video stream. For future work, the IoU and mAP of the model can be calculated to help evaluate the performance of the model and determine if get the weights from a certain Early Stopping Point is necessary. Furthermore, the influence of batch size in training and network-resolution in detection can also be tested.

References

[1] ACFR Orchard Fruit Dataset, the Australian Centre for Field Robotics, The University of Sydney, Australia, 2016, http://data.acfr.usyd.edu.au/ag/treecrops/2016-multifruit/

[2] S. Bargoti and J. Underwood, "Deep fruit detection in orchards," *2017 IEEE International Conference on Robotics and Automation (ICRA)*, Singapore, 2017, pp. 3626-3633.

[3] Bargoti, S. and Underwood, J. P. (2017), Image Segmentation for Fruit Detection and Yield Estimation in Apple Orchards. J. Field Robotics, 34: 1039–1060.

[4] McCool, C.; Sa, I.; Dayoub, F.; Lehnert, C.; Perez, T.; Upcroft, B. Visual Detection of Occluded Crop: For automated harvesting. In Proceedings of the International Conference on Robotics and Automation, Stockholm, Sweden, 16–21 May 2016.

[5] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi, “You Only Look Once: Unified, Real-Time Object Detection,” *the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, 2016, pp. 779-788.