IOTIOT

Internship Report

Active shelf monitoring for retail using YOLOv5

SEPTEMBER - DECEMBER 2020

Saurabh Prakash Giri, Mansi Meena

IT, IIIT Gwalior

Retail Store Item Detection using YOLOv5

The application of the new version of YOLO is introduced here, i.e. YOLOv5, to classify items on the shelf in a retail store. This program can be used to keep track of product inventory simply by using images of the products on the shelf.

Introduction

Object detection is a task for computer vision that involves the detection, localization and classification of object(s). In this task, we want our machine learning model first to tell if there is any object of interest in the photo. Draw a bounding box around the object(s) contained in the image, if any. The model must essentially categories the object defined by the bounding box. This role includes fast identification of objects so that they can be implemented in real-time. One of its main applications is its use of self-driving cars for real-time object detection.

YOLOv1, v2 and v3 models that perform real-time object detection were originally developed by Joseph Redmon, et al. YOLO "You Only Look Once" is a state-of-the-art algorithm for real-time deep learning used in images and videos for object detection, location and classification. This algorithm is very fast, precise and at the forefront of projects focused on object detection.

Each of the versions of YOLO continued to enhance the accuracy and efficiency of the previous one. Then YOLOv4 was produced by another team, adding more to model efficiency, and Glenn Jocher released the YOLOv5 model in June 2020. This model decreases the scale of the model dramatically (YOLOv4 on Darknet had 244MB size whereas YOLOv5 smallest model is of 27MB). YOLOv5 also claims a higher precision and more frames per second than YOLOv4, taken from the Roboflow.ai website, as seen in the graph below.

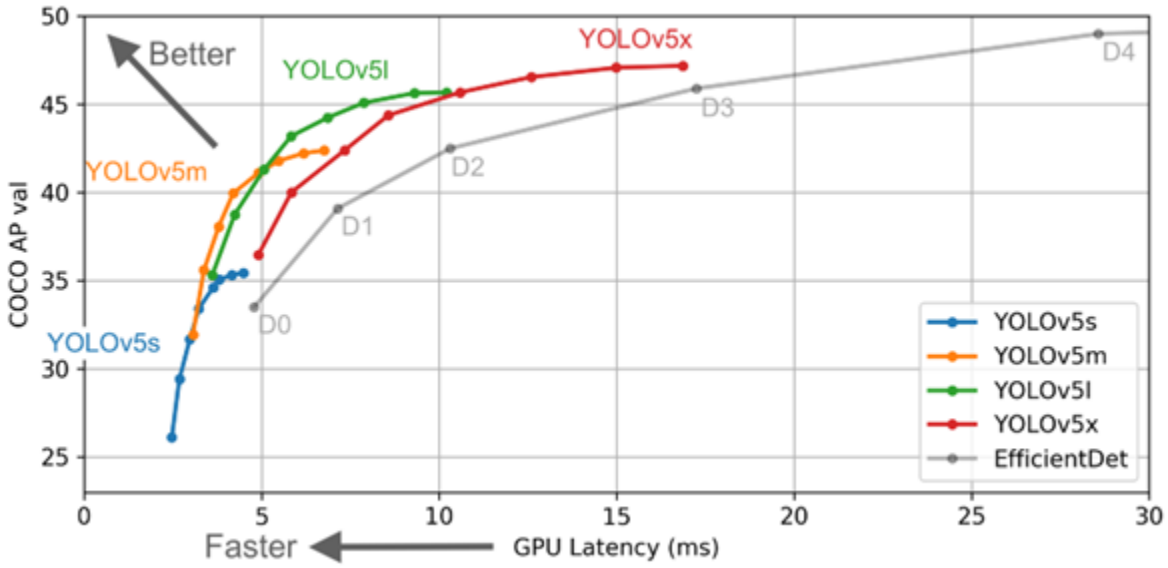


Fig 1.1: Comparison of YOLOv5 vs EfficientDetNet

In this, we will only focus on the use of YOLOv5 for retail item detection.

Objective

To use YOLOv5 to draw bounding boxes over retail products in pictures using SKU110k dataset.

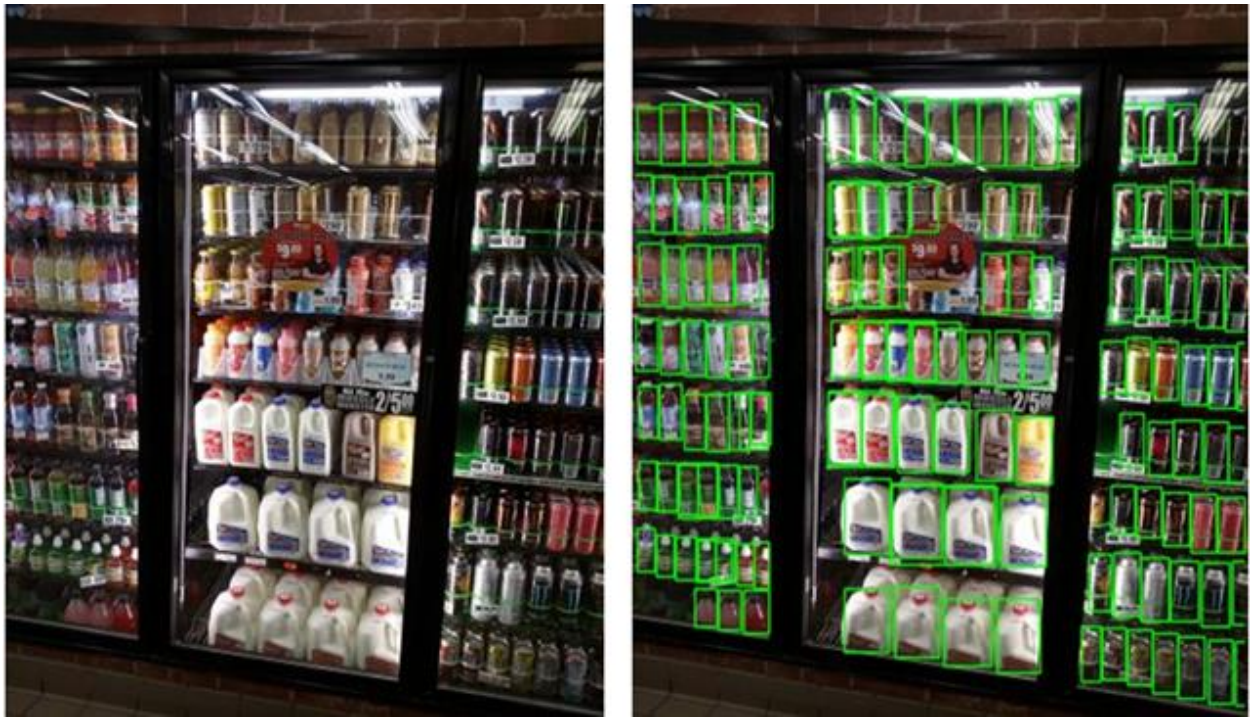


Fig 1.2: Store shelf image (on left) vs desired output with bounding box drawn on objects (right)

Dataset

To do this task, we first downloaded the SKU110k image dataset from the following link: http://trax-geometry.s3.amazonaws.com/cvpr_challenge/SKU110K_fixed.tar.gz. The SKU110k dataset is based on images of retail objects in a densely packed setting. It provides training, validation and test set images and the corresponding .csv files which contain information for bounding box locations of all objects in those images. The .csv files have object bounding box information written in the following columns:

image_name,x1, y1, x2, y2, class, image_width, image_height

where x1,y1 are top left co-ordinates of bounding box and x2, y2 are bottom right co-ordinates of bounding box, rest of parameters are self-explanatory. An example of parameters of train_0.jpg image for one bounding box, is shown below. There are several bounding boxes for each image, one box for each object.

train_0.jpg, 208, 537, 422, 814, object, 3024, 3024

In the SKU110k dataset, we have 2940 images in the test set, 8232 images in the train set and 587 images in the validation set. Each image can have varying number of objects, hence, varying number of bounding boxes.

Methodology

We only took 998 images from the training set from the dataset and went to the Roboflow.ai website, which offers an online image annotation service in various formats, including the format provided by YOLOv5. The explanation for only choosing 998 images from the training set is that the image annotation service of Roboflow.ai is free only for the first 1000 images.

Preprocessing

Preprocessing of images includes resizing them to 416x416x3. This is done on Roboflow's platform. An annotated, resized image is shown in figure below:

Fig 1.3: Image annotated by Roboflow

Automatic Annotation

The bounding box annotation .csv file and training set images are uploaded to the Roboflow.ai website, and Roboflow.ai's annotation service automatically draws bounding boxes on images using the annotations contained in the .csv files as seen in the above picture.

Data Generation

Roboflow also gives option to generate a dataset based on user defined split. I used 70–20–10 training-validation-test set split. After the data is generated on Roboflow, we get the original images as well as all bounding box locations for all annotated objects in a separate text file for each image, which is convenient. Finally, we get a link to download the generated data with label files. This link contains a key that is restricted to only your account and is not supposed to be shared.

Hardware Used

The model was tested on a Google Colab Pro notebook with a Tesla P100 16GB Graphics Card. It costs $9.99 and it's good for using it for a month. It is also possible to use Google Colab notebooks that are free, but there is restricted use of session time.

Code

The code is present in jupyter notebook in attached files. However, it is recommended to copy the whole code in Google Colab notebook.

It is originally trained for COCO dataset but can be tweaked for custom tasks which is what we did. We started by cloning YOLOv5 and installing the dependencies mentioned in requirements.txt file. Also, the model is built for Pytorch, so we import that.

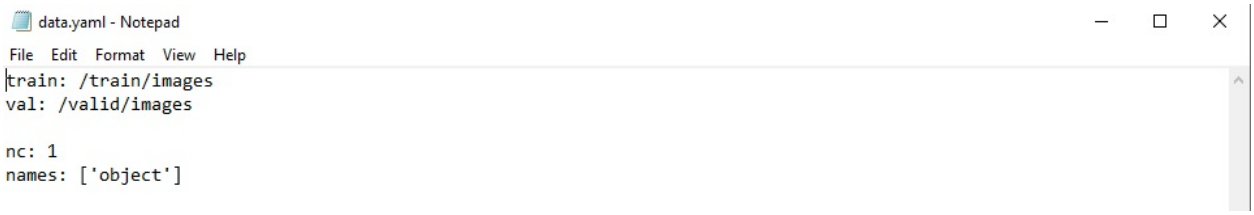
```
!git clone https://github.com/ultralytics/yolov5  # clone repo
!pip install -r yolov5/requirements.txt  # install dependencies
%cd yolov5

import torch
from IPython.display import Image, clear_output  # to display images
from utils.google_utils import gdrive_download  # to download models/datasets

clear_output()
print('Setup complete. Using torch %s %s' % (torch.__version__, torch.cuda.get_device_properties(0) if torch.cuda.is_available() else 'CPU'))
```

First, the dataset we generated at Roboflow.ai is downloaded. Training, test and validation sets and annotations can also be downloaded from the following code. It also generates a .yaml file containing training and validation set paths as well as what classes are present in our knowledge. If you use Roboflow for info, as it is special per user, don't forget to enter the key in the code.

This file tells the model the location path of training and validation set images alongwith the number of classes and the names of classes. For this task, number of classes is "1" and the name of class is "object" as we are only looking to predict bounding boxes. data.yaml file can be seen below:



Network Architecture

Next, let's describe the YOLOv5 network architecture. The design used by author Glenn Jocher for COCO dataset training is the same. On the network, we didn't change anything. In order to adjust bounding box size, colour and also to remove labels, however, few tweaks were required because of so many boxes, otherwise labels would jumble the picture. In detect.py and utils.py scripts, such tweaks have been produced. The network is stored as a file called custom yolov5.yaml.

Training

Now we start the training process. We defined the image size (img) to be 416x416, batch size 32 and the model is run for 300 epochs. If we don't define weights, they are initialized randomly.

```
# train yolov5s on custom data for 300 epochs
# time its performance
%%time
%cd /content/yolov5/
!python train.py --img 416 --batch 32 --epochs 300 --data '../data.yaml' --cfg ./models/custom_yolov5s.yaml --weights '' --name yolov5s_results --nosave --cache
```

It took 4 hours 37 minutes for training to complete on a Tesla P100 16GB GPU provided by Google Colab Pro. After the training is complete, model's weights are saved in Google drive as last_yolov5_results.pt

Observations

We can visualize important evaluation metrics after the model has been trained using the following code:

```
# we can also output some older school graphs if the tensor board isn't working for whatever reason...
from utils.utils import plot_results; plot_results()  # plot results.txt as results.png
Image(filename='./results.png', width=1000)  # view results.png
```

The following 3 parameters are commonly used for object detection tasks: ·

- **GIoU** is the Generalized Intersection over Union which tells how close to the ground truth our bounding box is.
- **Objectness** shows the probability that an object exists in an image. Here it is used as loss function.
- **mAP** is the mean Average Precision telling how correct are our bounding box predictions on average. It is area under curve of precision-recall curve.

It is seen that Generalized Intersection over Union (GIoU) loss and objectness loss decrease both for training and validation.

Mean Average Precision (mAP) however is at 0.7 for bounding box IoU threshold of 0.5.

Recall stands at 0.8 as shown below:

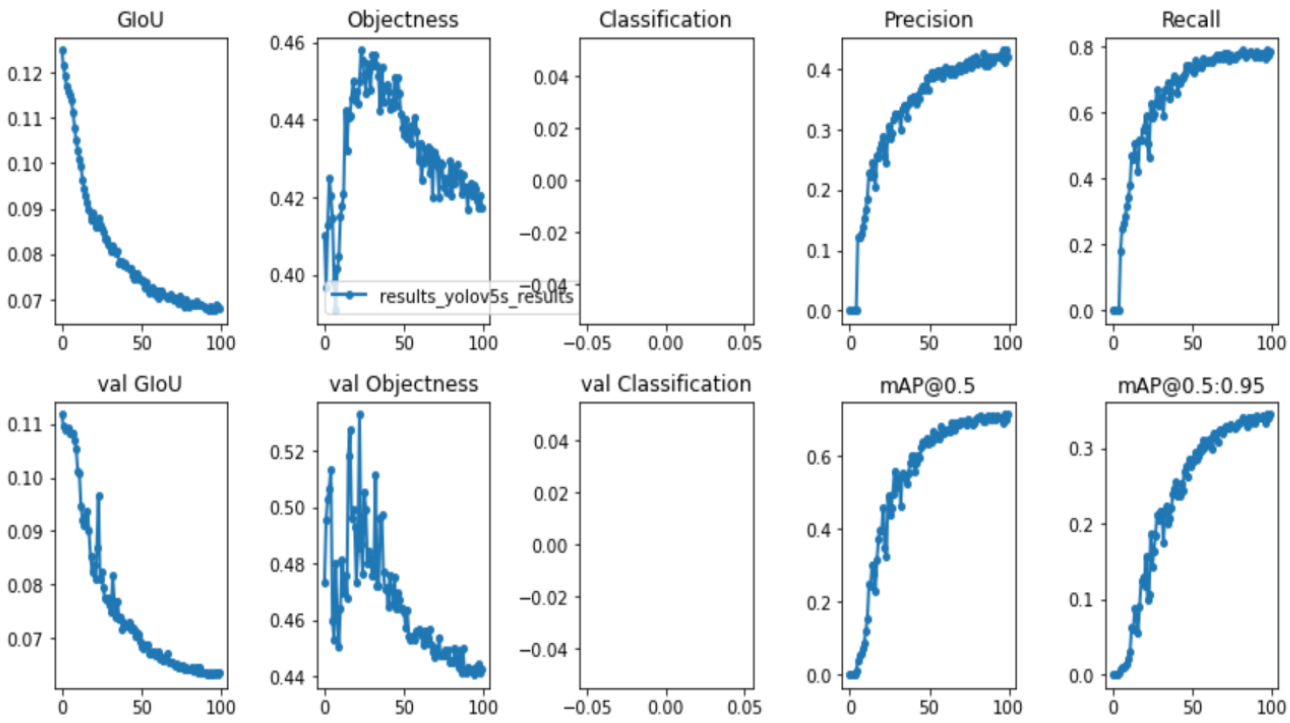
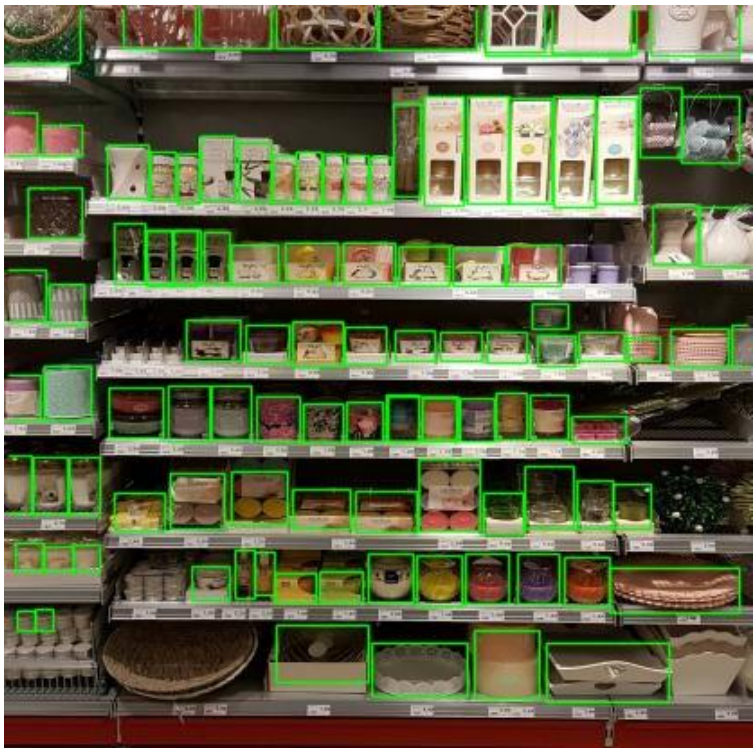


Fig 1.4: Observations of important parameters of model training

Results

Following images show the result of our YOLOv5 algorithm trained to draw bounding boxes on objects. The results are pretty good.



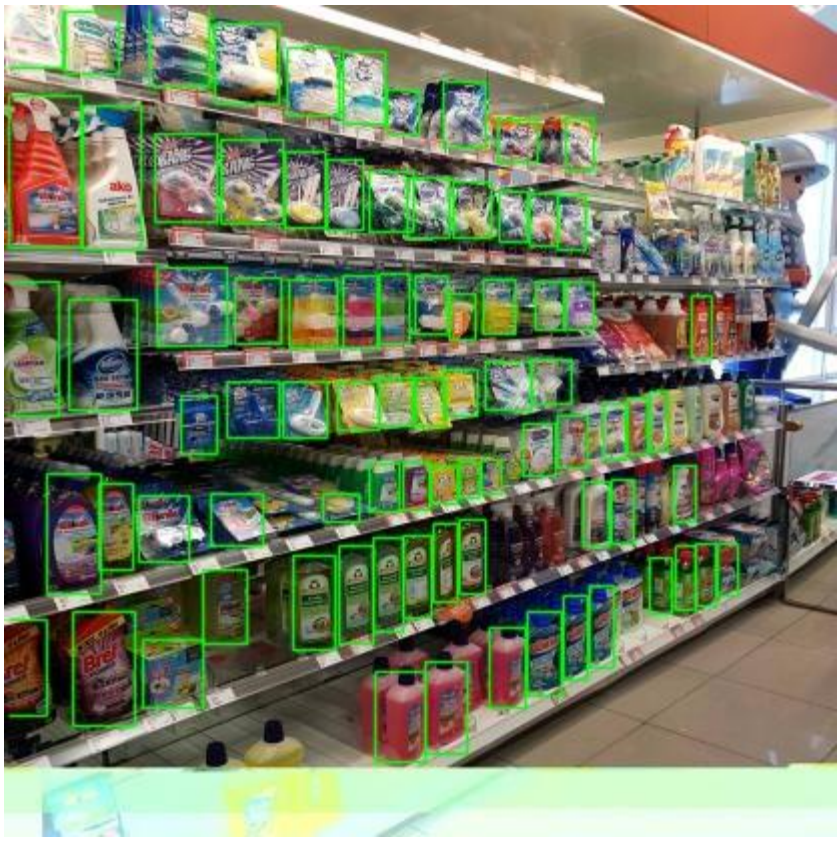


Fig 1.5: Original test set image (on left) and bounding boxes drawn images by YOLOv5 (on right)

Conclusion

Controversies aside, YOLOv5 performs well and can be customized to suit our needs. However, training the model can take significant GPU power and time. It is recommended to use atleast Google Colab with 16GB GPU or preferably a TPU to speed up the process for training the large dataset.

This retail object detector application can be used to keep track of store shelf inventory or for a smart store concept where people pick stuff and get automatically charged for it. YOLOv5's small weight size and good frame rate will pave its way to be first choice for embedded-system based real-time object detection tasks.

References

[1] https://roboflow.com/

[2] RoboflowArticle: https://blog.roboflow.com/training-a-tensorflow-faster-r-cnn-object-detection-model-on-you-own-dataset/

[3] RCNNs: https://towardsdatascience.com/deep-learning-for-object-detection-a-comprehensive-review-73930816d8d9

[4] Ren, S., He, K., Girshick, R., & Sun, J. (2016). Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE transactions on pattern analysis and machine intelligence*, *39*(6), 1137-1149.