Car Detection using YOLO algorithm

Prajwal Kalpande
Autonomous Driving-Car Detection
Winter In Data Science
Analytics Club,IITB
prajwalkalpande3@gmail.com

***Abstract*—Deep learning is being increasingly used for real-time object analysis. Compared to other traditional two-stage object detection algorithms YOLO is extremely fast and accurate. I have summarized about the working of YOLO and its CNN Architrecture in this report. I used a pre-trained YOLO model for detecting objects in the images from car detection dataset containing sequential images from the video captured through cameras in my project. I also implemented Non-Max Suppression using IoU to predict accurate bounding boxes and class probabilities in this project. Further, fine-tuning was done on a different dataset to achieve better results.**

***Index Terms*—YOLO; Car Detection; Deep Learning; Convolutional Neural Network; Object Detection**

I. Introduction

Autonomous driving is a hot topic in the area of artificial intelligence and machine learning where numerous researches are being conducted to make driver-less vehicles mainstream. The detection of vehicles ahead and the traffic conditions while driving are important factors for safe driving. This brings the need for an accurate and fast car detection system. The problem of object detection is more complex than classification, which not only recognizes what the object is but also locates it in the image. The use of deep convolutional networks (CNNs) has achieved amazing success in the field of vehicle object detection. CNNs have a strong ability to learn image features and can perform multiple related tasks, such as classification and bounding box regression. The two-stage methods like Region-CNN (R-CNN) generate a candidate box of the object via various algorithms and then classify the object by a convolutional neural network. However, the one-stage method does not generate a candidate box but directly converts the positioning problem of the object bounding box into a regression problem for processing. YOLO :You Only Look Once is a single-stage Unified, Real-Time Object Detection algorithm which simultaneously predicts the bounding boxes along with their labels. As a result, it is 1000x faster than R-CNN and achieves high accuracy while also being able to run in real time.

YOLO algorithm applies a single neural network to the full image, and then divides the image into regions and predicts bounding boxes and probabilities for each region. These bounding boxes are weighted by the predicted probabilities. The algorithm "only looks once" at the image in the sense that it requires only one forward propagation pass through the neural network to make predictions. After non-max suppression (which makes sure the object detection algorithm only detects each object once), it then outputs recognized objects together with the bounding boxes.

Besides being fast, YOLO learns generalizable representations of objects so that when trained on natural images and tested on new data, the algorithm outperforms other top detection methods. Although, YOLO eliminates candidate region proposals, thereby greatly improving the detection speed, the simple estimation of the position of the object ignores the information of many small objects and dense objects, reducing the overall detection accuracy.

II. Background and Prior Work

[1] The first YOLO model was introduced by Joseph Redmon et all in their 2016 paper titled "You Only Look Once: Unified, Real-Time Object Detection". Till that time RCNN models were the most sought-after models for object detection. Although the RCNN family of models were accurate but were relatively slow. YOLO was created with the goal of making it easy to optimize, real-time and allowing end to end training. YOLOv1 sported a 63.4 mAP with an inference speed of 45 frames per second (22ms per image). At that time, it was a huge improvement of speed over the RCNN family for which inference rates ranged from 143ms to 20 seconds. The subsequent versions were published by them in 2016, 2017 and by Alexey Bochkovskiy in 2020.

III. Brief Overview of YOLO

YOLO divides the input image into an $S \times S$ grid. Each grid cell predicts$B$ bounding boxes and confidence scores for those boxes. The confidence scores are a measure of how sure the model is that the box contains an object and also how accurate it thinks its prediction is. Thus, confidence is defined as $P_r(Object) * IOU_{pred}^{truth}$, where IOU_{pred}^{truth} stands for Intersection Over Union of the predicted box and the ground truth(actual box). Each bounding box consists of 5 predictions: $x, y, w, h,$ and a confidence score. The center of the bounding box with respect to the grid cell is denoted by the coordinates (x, y). The width w and height h of the bounding box are predicted as a fraction of the width and height of the whole image. As a result, x, y, w, h all lie in the range $[0, 1]$. YOLO only predicts one set of C conditional class probabilities ($P_r(Class_i|Object)$) per grid cell even though

the grid cell has B bounding boxes. Thus, for each grid cell, YOLO predicts $C + B$x 5 parameters. Total prediction tensor for an image = S x S x ($C + B$ x 5) Finally, YOLOv1 applies

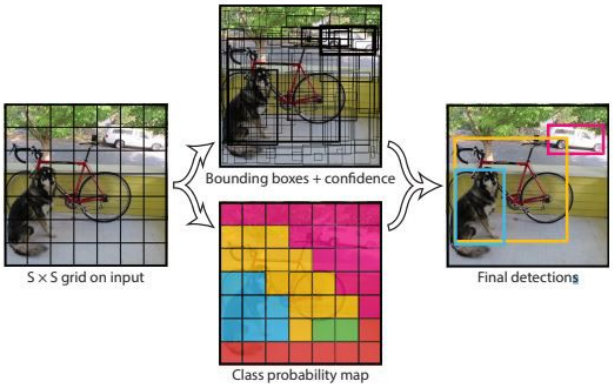

Fig. 1. YOLOv1 conceptual design [1]

Non Maximum Suppression (NMS) and thresholding to report final predictions.

A. YOLOv1 CNN Design

The CNN architecutre of YOLOv1 is depicted in the Figure 2. It has 24 convolution layers that act as a feature

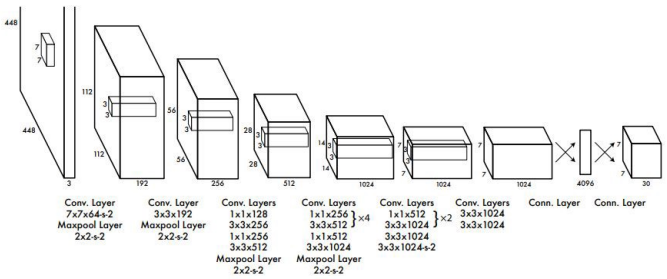

Fig. 2. YOLOv1 CNN Architecture [1]

extractor. They are followed by 2 fully connected layers that are responsible for classification of objects and regression of bounding boxes. This network architecture was inspired by the GoogLeNet model for image classification. Instead of the inception modules used by GoogLeNet, they used 1 × 1 reduction layers followed by 3 × 3 convolutional layers. Leaky ReLU activation is used for all layers except the final layer. The final layer uses a linear activation function. The Leaky ReLU activation function used is as follows:

$$f(x) = \begin{cases} x & if x > 0 \\ 0.1x & otherwise \end{cases}$$

The input to the network is a batch of images and the output is a list of bounding boxes along with the recognized classes. For PASCAL VOC dataset, YOLOv1 uses S = 7, B = 2 and C = 20. Thus, final YOLO prediction for PASCAL VOC was a 7 x 7 x (20 + 5 x 2) = 7 x 7 x 30 tensor.

YOLO increases the weight (λ coord = 5)for predictions from bounding boxes containing objects and reduces the weight λ noobj = 0.5) for bounding boxes that do not contain any objects to prevent overpowering of boxes containing no object since they are usually present in large nuumbers. The loss function which was optimized during training is as follows:

$$\begin{aligned}
&\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
&+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2
\end{aligned}$$

Fig. 3. Loss Function [1]

B. Implemented Functions

In this project, I implemented some functions used in YOLO using Tensorflow and Keras in Python to understand their working. IOU was used in YOLOv1 whereas the concept of Anchor Boxes was introduced in YOLOv2 [2]. Anchor boxes allow grid cells to detect more than one object. They are chosen by exploring the training data to choose reasonable height/width ratios that represent the different classes. [2] The original paper used k-means clustering based on IOU instead of Euclidean Distance on all bounding boxes for various values of k to get the anchor boxes. Thus, the final output tensor is S x S x No. of anchor boxes x ($C + B$ x 5). The functions which I implemented are summarized as follows:

1) IOU: Intersection over Union is an evaluation metric used to measure the accuracy of an object detector which predicts bounding boxes on a particular dataset. It is calculated using the following formula :

$$\frac{\text{Area of overlap between Ground Truth and Bounding Box}}{\text{Area of union of Ground Truth and Bounding Box}}$$

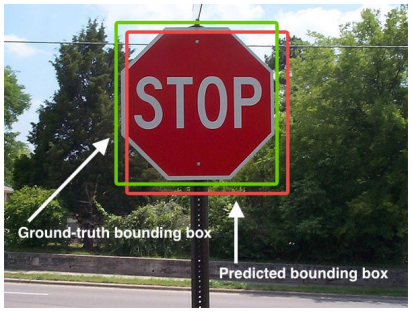

Fig. 4. IOU

2) Non Max Suppression: The "class" scores are obtained by multiplying confidence scores with $P_r(Class_i|Object)$. Based on them, thresholding is done to filter out the boxes. However, the number of bounding boxes obtained after this step are too many.

To tackle this, Non Max Suppression(NMS) is used. It makes

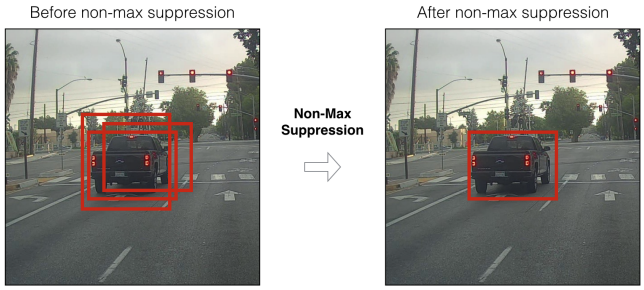

Fig. 5. NMS

use of IOU to filter out the better bounding boxes and get rid of the rest. NMS first selects the box that has the highest score. Then, it computes the overlap of this box with all other boxes, and removes boxes that overlap significantly ($IOU >= IOU_{threshold}$). It keeps repeating above steps until there are no more boxes with $IOU >= IOU_{threshold}$.

C. mean Average Precision (mAP)

Mean average precision (mAP) is a metric used to evaluate object detection models like R-CNN and YOLO. The mAP compares the ground-truth bounding box to the detected box and returns a score. The higher the score, the more accurate the model is in its detections. It is the mean of the Average Precision(AP) over all the classes. To calculate AP we first need to understand Confusion Matrix, Precision, Recall and Precision-Recall Curve.

1) Confusion Matrix: It is a performance measurement for machine learning classification problems where output can be two or more classes. It is a table with 4 different combinations of predicted and actual values.

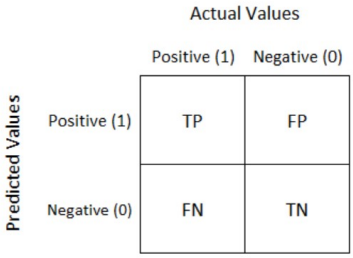

Fig. 6. Confusion Matrix

2) Precision: Precision measures how accurate your predictions are. i.e. the percentage of your predictions are correct.It measures how many of the predictions that your model made were actually correct.

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

3) Recall/Sensitivity: Recall measures how well you find all the positives. It is the proportion of TP out of the possible positives.

$$Precision = \frac{TP}{TP + FN} \tag{2}$$

4) Precision-Recall Curve: The precision-recall curve is used for evaluating the performance of binary classification algorithms. It provides a graphical representation of a classifier's performance across many thresholds rather than just a single value. AP is the weighted sum of precisions at each

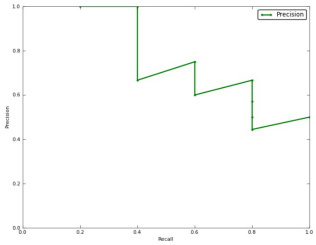

Fig. 7. [4]Precision-Recall Curve

threshold where the weight is the increase in recall from the previous threshold. In other words, AP is the area under the Precision-Recall curve.

$$AP = \int_0^1 p(r)dr \tag{3}$$

Nowadays, AP is calculated as the area under the interpolated Precision-Recall Curve. Thus, it can be redefined as follows:

$$AP = \sum (r_{n+1} - r_n) p_{interp}(r_{n+1}) \tag{4}$$

p_{interp} is the precision value for recall r with the maximum precision for any recall $\geq r$. These points are summed up in the following figure where Green Curve is $p_{interp}(r)$: In

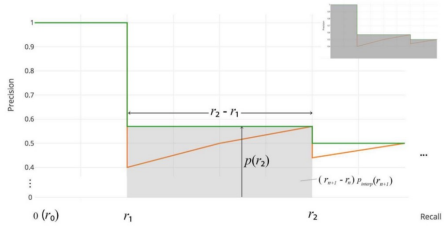

Fig. 8. [4]Interpolated Precision-Recall Curve

COCO mAP, a 101-point interpolated AP definition is used in the calculation. For COCO, AP is the average over multiple IoU (the minimum IoU to consider a positive match)

IV. Datasets

Two datasets were used in this project. The Car Object Detection dataset was used for performing inference i.e. just for testing the model.The dataset contains media of cars in all views.

Fig. 9. [6]Images used for Testing

The dataset used for fine-tuning was the Vehicles-OpenImages Dataset available in the public datasets of Roboflow. This dataset contains 627 images of various vehicle classes for object detection. These images are derived from the Open Images open source computer vision datasets.

Fig. 10. [7]Images used for Fine-Tuning

V. Detection Results using PreTrained Model

[5]The pretrained YOLOv5 model was used for doing inference and detecting objects on Car Object Detection dataset. The YOLOv5 model has a high inference speed of 140 FPS and unlike previous versions of YOLO, it is implemented in PyTorch. There are various model sizes available such as YOLOv5s (smallest), YOLOv5m, YOLOv5l, YOLOv5x (largest). Since, we just had to perform a inference on a small dataset, for the sake of simplicity YOLOv5s was used. The images obtained after detection as given in Fig. 11

VI. Detection results using Fine-tuned model

For fine-tuning YOLOv4-tiny [8], which is the compressed version of YOLOv4 was used. It is usually used for faster training and detection as it is 8 times faster than YOLOv4. However, its accuracy is $\frac{2}{3}$rd of that of YOLOv4. But in real-time vehicle/object detection faster inference time is more important than precision or accuracy, hence it has been used.

As YOLOv4 is DarkNet based the dataset was exported in YOLO DarkNet Format. After fine-tuning the network, the final mAP score obtained was 55.58 % with IOU threshold

Fig. 11. Detection Results of Pre-trained model

set to 0.5. The conf_threshold (threshold used for filtering boxes by confidence) used was 0.25. After training for 10,000 iterations, the final classification metrics obtained were as follows: 1) Precision = 0.57 2) Recall = 0.52 and 3) F1-score = 0.55.

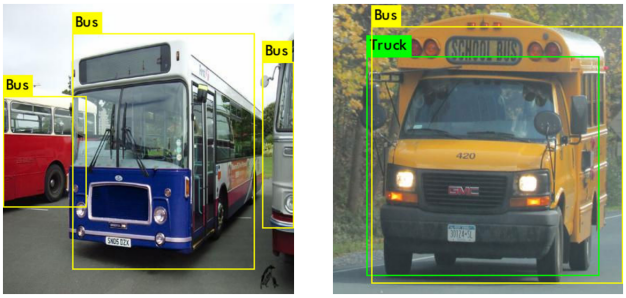

Fig. 12. Detection Results of Fine-tuned model

Interestingly if we neglect the top of the school bus it looks quite similar to a truck and so both truck and bus have been detected by the model. Such errors can be avoided by training on more data and using larger model size.

VII. Links to colab notebooks

A. Inference using Pre-trained Model

B. Fine-tuning YOLOv4-tiny modell

Acknowledgment

I would like to thank my mentor Valay Bundele for guiding me through the project and Analytics Club, IIT Bombay for organizing Winter in Data Science and giving me an opportunity to explore the use of data science in autonomous driving.

References

[1] Joseph Redmon, S. Divvala, R. Girshick, Ali Farhadi. "You Only Look Once: Unified, Real-Time Object Detection", IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, arxiv.org

[2] Redmon, Joseph, and Ali Farhadi. "YOLO9000: Better, Faster, Stronger", 2016, arxiv.org

[3] Joseph Redmon, Ali Farhadi. "YOLOv3: An Incremental Improvement", 2018, arxiv.org

[4] Hui, Jonathan. "MAP (Mean Average Precision) for Object Detection." Medium, 3 Apr. 2019, jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173

[5] https://github.com/ultralytics/yolov5

[6] https://www.kaggle.com/sshikamaru/car-object-detection

[7] https://public.roboflow.com/

[8] https://models.roboflow.com/object-detection/yolov4-pytorch