Comparing Object Detection Algorithms

Vavruska, Ryan
Computer Science
University of Massachusetts - Lowell
Lowell, USA
ryan_vavruska@student.uml.edu

***Abstract*—In this paper, different object detection algorithms are discussed to find out which modern-day algorithm is best suited for object detection tasks. The three algorithms that are being compared are YoloV5, SSD and FasterRCNN. Pre-trained versions of these algorithms are taken, and then fine-tuned using google's open images v6 dataset to see how well these algorithms perform when adapting to foreign data. In this paper, these algorithms will be explained, with the differences between them being discussed. It will then be concluded which one stands out as being the best algorithm today.**

I. Introduction

With the advancements in object detection over the years, companies are looking at more and more ways to automate processes. From self-driving cars to security systems, there are many ways that object detection algorithms can be used to take out human interaction from everyday tasks. With advancements in object detection comes a multitude of implementations that all accomplish roughly the same task. In this paper, I will discuss three different object detection algorithms, Faster RCNN, Single Shot MultiBox Detector (SSD) and You Only Look Once (YOLOv5), comparing the performance between the three systems to hypothesize which algorithm would best be suited for a real-time system, like a security system. This work is connected to previous work in the field, as new versions of these algorithms are constantly being released, as well as novel algorithms as well. With this being the case, there is a constant need to re-evaluate how these algorithms perform, to ensure that the industry is constantly evolving to solve their problem using the best now available algorithm.

There are many applications of object detection algorithms that require fast and accurate results. These applications include systems that often provide quick info about an image or frame of a video. An example of this is a security system. Many businesses around the world run 24/7 surveillance systems. When paired with an object detection algorithm, the system could feed frames of the surveillance system into the trained model, and mark timestamps in the video feed if the model with high confidence spotted a human in the feed during night-time hours. Another example that a system like this can solve is using object detection in self-driving cars. Self-driving cars typically work with a multitude of cameras that are placed around the car that are constantly monitoring the environment around the vehicle. Paired with other types of algorithms, object detection can be used to determine if a person or another vehicle is in frame, to help impact the decision-making process of the vehicle, ensuring that the self-driving car does not hit something while it is in motion.

II. Related Work

There has been a lot of research in the field, with people often pondering which object detection algorithm to use, with the best option changing rapidly as the area gets more and more attention.

In 2006, Nascimento et al. did an analysis on leading object detection algorithms at the time. Their analysis was primarily focused on surveillance footage and then compared BBS, W4, SGM, MGM and LOTS, with LOTS proving to be the most accurate at the time the article was published. In their analysis they also did complexity analysis of the five different object detection models. They found that MGM was the most complex and had the highest computational cost. Looking at Nascimento et al's work, there were researchers striving for higher performance in object detection back before object detection was as advanced as it is today.

In 2017, Noman et al, did an analysis on Microsoft Azure Cloud object detection and Google TensorFlow object detection. In their report they detail an analysis between the two different platforms and determine that the performance between the two is close. With a difference of 3.34%, they show that object detection methods just keep getting better and better, with a constant desire to improve upon their predecessors. With the similarity between the two platforms, one can then look to choosing their platform based on their needs. Whether cloud or local data processing is desirable, as well as the method of training can be looked at when it comes to these two systems.

In 2021, Srivastava et. al did a similar analysis of the data where they tested FasterRCNN, Yolov3 and SSD using the COCO dataset. Their findings showed that at the time Yolo-v3 was the best option for providing real-time analysis with SSD falling close-second behind, and FasterRCNN performing poorly compared to the other two. While quantitatively Faster RCNN had a higher accuracy that SSD in their findings, the performance gains from using SSD over Faster RCNN put SSD in a top spot according to Srivastava et al.

Looking at related works to this topic, there is an unknown that is constantly shifting as more and more technologies develop. The question as to what the highest performing object detection algorithm will never truly be answered; but, analysis

can continue to be done so that industries can make the best choice from the selection that is available today.

III. Methodologies

A. YOLOv5

One of the most precise and accurate image algorithms available. Built on a customized architecture named Darknet. It works by diving images into a grid system. Within this grid system, each grid is then responsible for detecting objects within itself. The network is split up into three main pieces. The first been the backbone, which is a conventional neural network that aggregates and forms image features. The second feature is the neck, which mixes and combines image features to pass them to prediction. The final major piece is the head, which consumes the features from the neck and takes the prediction step. All of these come together to form one of the fastest and best performing object detection algorithms that exist today.

B. SSD

A much faster object detection model compared to other similar models. SSD heavily relies on the generation of bounding boxes and the extraction of feature maps in the input images. It is built on a feed forward complex network that builds an amount of standard-size bounding boxes, and for each object in those boxes they are then given a score. These scores are then evaluated to give a confidence level based on the score that is given.

C. FasterRCNN

R-CNN stands for Region-based Convolutional Neural Network. Combines region proposals for object segmentation and high capacity CNNs for object detection. It uses a selective search algorithm, and proposes several candidate regions, wrapping these proposals and extracting distinct features. These features are then fed into an SVM for recognizing objects of interest in the proposal regions. Faster RCNN allows the proposals to be learned by the network, eliminating the need for the selective search algorithm for generating the proposals.

IV. Proposed Approach

A. Data set

Fig. 1: FiftyOne Dataset Viewer

All models were trained on Google Cloud Platform using a T4 GPU. The data set that was used for comparison of the models was a subset of the Open-Images v6 dataset published by Google. In order to keep things simple and keep down training times, the subset consisted of just images that contained a single class, which was cars. The introduction of multiple classes could easily be implemented by adding more training data, but in order to prioritize the focus on the performance of these object detection algorithms, the dataset was focused on cars. The dataset was obtained in one form using the officially supported fiftyone downloader, in which the set was pruned to including cars as a single class. Another form of the dataset was obtained using Roboflow, which offers similar data in specific formats that can be fed into certain models directly. With the following dataset, I was able to use separate implementations of Yolov5, SSD and FasterRCNN to test the performance of each model on a similar dataset.

The dataset consists of a multitude of jpg images containing cars, with annotation files attached. These files contain bounding box information, which is fed into the model to determine which object in the image represents a car.

B. YOLOv5

For Yolov5, I used the Ultralytics implementation of Yolov5 and fed the Roboflow dataset into it. With the Ultralytics implementation of Yolov5 I was able to test the accuracy of the model in multiple different configurations to see which yielded the best results. In all the configurations of Yolov5 I used the small version of their pre-trained model which is pre-trained on the coco dataset. Using this, I was able to try multiple different configurations of freezing different layers to see which ones produced the best results. The first configuration that was tested was one where all layers, except for the output layer were frozen. The next configuration tested was an instance where only the first ten layers, which consists of the backbone of the model were frozen. Finally, a configuration of Yolov5 was tested where no layers were frozen and instead all the weights were changed, using the pretrained model as a starting point for all configurations. All configurations were trained for 50 epochs.

C. SSD

For the analysis of Single Shot MultiBox Detection (SSD), a PyTorch implementation of the algorithm was used. This implementation was then modified to read the data in a VOC format that was exported from Roboflow. Some changes to the model were also made to ensure that the loss was calculated correctly and that the model would work properly on a smaller dataset. The model was then trained for 5000 iterations to ensure that the model robust on all the different types of cars in the training set.

D. FasterRCNN

For FasterRCNN I used a pretrained version of PyTorch's fasterrcnn_resnet50_fpn. With this pre-trained model, I took the dataset obtained using fiftyone's downloader and pruned

the classification list to only include images that were included in the download train split. I then took the input data and ensured it was in the correct format for the model, before training the model on the open-images input data. After all the models were trained, the model was then run on a validation dataset, with the precision and accuracy of the model being recorded so that F1 scores could be calculated. The F1 scores of these models were then compared to determine which model yielded the best performance of the three object detection algorithms.

V. Experimental Results and Discussion

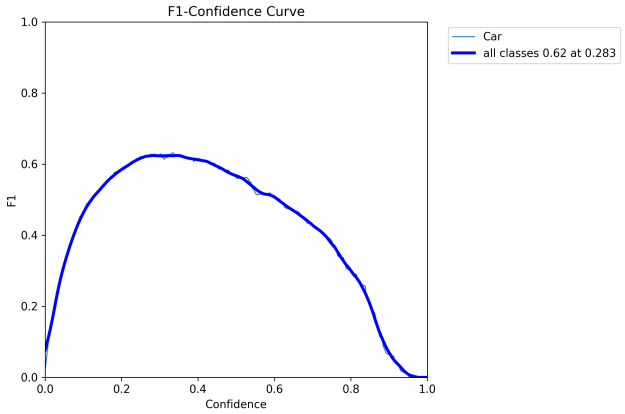

Fig. 2: All Layers Trained

Figure 2 is a figure showing the F1 scores that YOLO observed after training all layers of the network. Weights were initialized on the entire network from the pre-trained COCO model, with no layers being frozen and the entire network updated with the car dataset. It can be observed that the F1 score peaked at 0.62 with a confidence of 0.283.

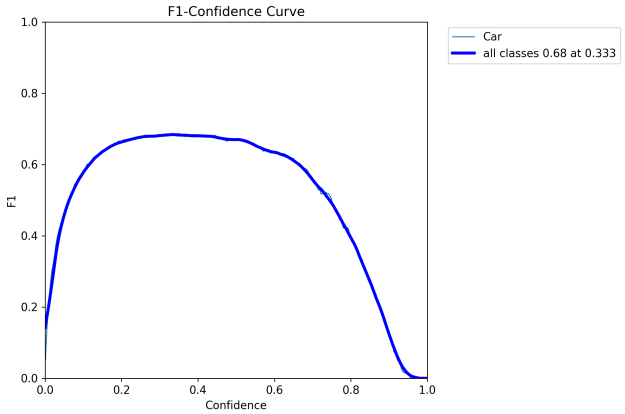

Fig. 3: Last 14 Layers Trained

Figure 3 shows the F1-confidence curve when the network is initialized with the COCO dataset weights, and then the first 10 layers, which consists of the backbone of the network is frozen. It can be observed that the curve is much more straight, meaning the confidence and accuracy of the network is much higher than when the entire network was fine-tuned on the dataset. In this iteration, the F1 score peaks at 0.69 with a confidence of 0.621, however looking at the curve it can be noted that the network maintains this level of F1 score as the confidence goes up and down.

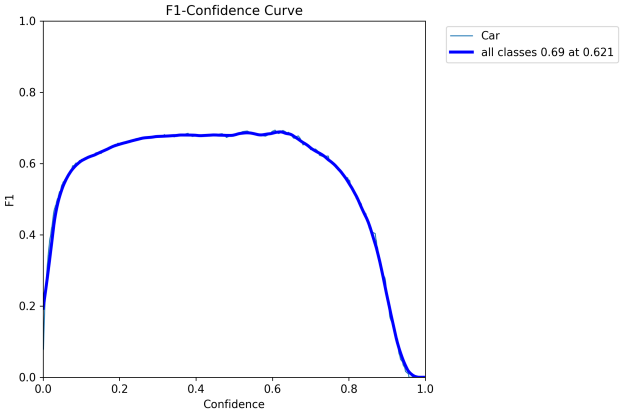

Fig. 4: Output Layer Trained

Looking at figure 4, which shows the F1 confidence curve of the entire network other than the last output layer is frozen. The network is then trained on the open-image dataset with the confidence curve peaking at 0.62 F1 score and a confidence of 0.283.

When looking at all of the implementations of YOLO, it can be observed that the implementation where the backbone of the network is frozen and the rest of the layers are allowed to be tuned on the image dataset proved to perform the best overall, while having the highest F1 scores and confidence levels when compared to their counterparts. As the version with the backbone frozen performed the best overall, its AP value of 0.597 is taken as a means of comparing its performance.

The next algorithm that was observed and analyzed was SSD, as can be seen in figure 6, which is the output of an image that was fed-into the fine-tuned model, SSD provided mostly accurate results. When trained on just the car data-set SSD provided an AP value of 0.498. This is just slightly lower than that of Yolov5, the model providing fairly accurate results when it comes to predicting cars.

The final CNN type that was analyzed was Faster RCNN, which when compared to the other algorithms is the lowest performing. My implementation of RCNN had very poor results and was unable to train properly to the data given to it. As can be seen in 7 the box prediction of the Faster RCNN implementation was extremely poor and the model was unable to properly distinguish between different objects. As can be seen in the image, the model appears to draw a box around most objects in the input image, but is unable to throw out the objects that are not cars in its resulting output.

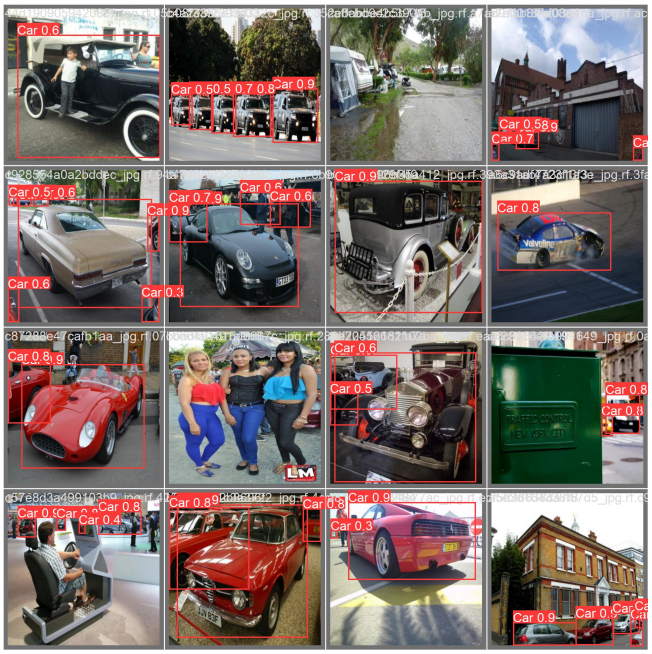

Fig. 5: YOLO Prediction Map

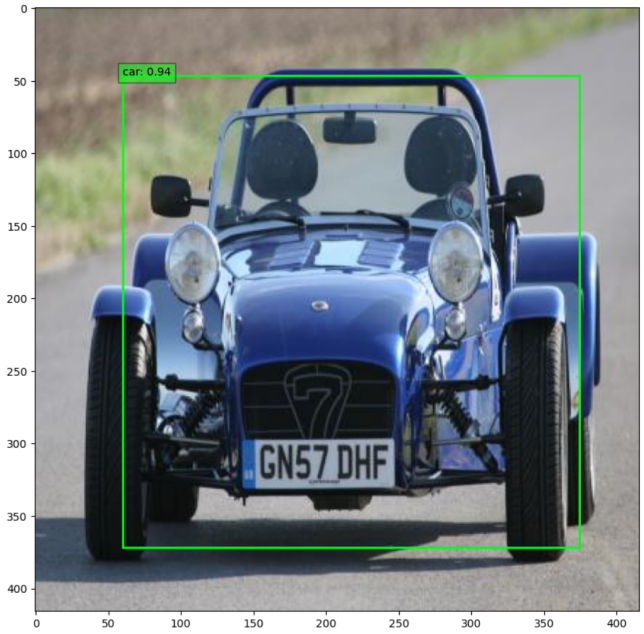

Fig. 6: SSD Prediction

Algorithm Name	AP
Yolov5	0.597
SSD	0.498
Faster RCNN	-

When comparing the accuracy between the three networks it can be seen that YOLOv5 provides the highest accuracy and confidence out of the three algorithms that were tested. Second comes SSD which offers a medium of speed and accuracy, with the second-quickest training of the three implementations

Fig. 7: Faster RCNN Prediction

as well as the second-most accurate. Finally, it can be observed that Faster RCNN suffers the most, being the least accurate of the three. However, while Faster RCNN might not shine in accuracy, it does appear to be the most lightweight of the three implementations in terms of training.

VI. Future Work

Some future work that could be added to this project would be the inclusion of more classes. While this paper only focuses on a single class, distinguishing between cars are background, open images include over 500 different classes to choose from. Work could be done to include all the classes so that the detection system is much more robust. Another improvement that could be made, would be to include more data overall. The network was primarily trained on a subset of data of roughly 1000 images for each algorithm. With more data, the network could become more exposed to different object and thus, provide more accurate results. Finally, the implementation of Faster RCNN could be corrected to more accurately predict bounding boxes for cars. As it is now, the training loop could be more fine-tuned to more correctly predict cars in input images.

VII. Conclusion

This paper compared different CNN-based object detection algorithms to determine which algorithm performs the best overall. With this comparison, it can be shown in the small amount of data that was provided to these algorithms using pre-trained models as a base, that YoloV5 performs well above the others when compared to SSD and Faster RCNN. In terms of accuracy and performance, YOLO outshines the other two in every aspect. With all networks being trained used Google Open-Images-V6, meaning that the baseline for all these trainings were the same, it can be assumed that each network existed on a level playing field. Coming in second, SSD outshines Faster RCNN, coming in second place among the three algorithms. SSD provides decent accuracy with a much higher level of speed when compared to Faster RCNN. Finally Faster RCNN shows that its performance really does not match those of its more modern implementations and

leaves much to be desired. Back to the original question, of which algorithm would perform best when implemented in a system such as a security system, it can be shown that YOLOv5 would give the best results.

REFERENCES

[1] J.C. Nascimento and J.S. Marques. "Performance evaluation of object detection algorithms for video surveillance". In: *IEEE Transactions on Multimedia* 8.4 (2006), pp. 761–774. DOI: 10.1109/TMM.2006.876287.

[2] Wei Liu et al. "SSD: Single Shot MultiBox Detector". In: *Computer Vision – ECCV 2016*. Ed. by Bastian Leibe et al. Cham: Springer International Publishing, 2016, pp. 21–37. ISBN: 978-3-319-46448-0.

[3] Mohammed Noman, Vladimir Stankovic, and Ayman Tawfik. "Object Detection Techniques: Overview and Performance Comparison". In: *2019 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT)*. 2019, pp. 1–5. DOI: 10.1109/ISSPIT47144.2019.9001879.

[4] Glenn Jocher. *YOLOv5 by Ultralytics*. Version 7.0. May 2020. DOI: 10.5281/zenodo.3908559. URL: https://github.com/ultralytics/yolov5.

[5] Alina Kuznetsova et al. "The Open Images Dataset V4: Unified image classification, object detection, and visual relationship detection at scale". In: *IJCV* (2020).

[6] Shrey Srivast et al. *Comparative Analysis of Deep Learning Image Detection Algorithms*. Dec. 2020. DOI: 10.21203/rs.3.rs-132774/v1.

[7] Raza Rizwan. *google-open-image-cars-dataset Dataset*. https://universe.roboflow.com/raza-rizwan/google-open-image-cars-dataset. Open Source Dataset. visited on 2022-12-05. July 2022. URL: https://universe.roboflow.com/raza-rizwan/google-open-image-cars-dataset.

[8] Max deGroot. *SSD: Single Shot MultiBox Object Detector, in PyTorch*. URL: https://github.com/amdegroot/ssd.pytorch.