Analyzing Most Popular Object Detection Models for Deep Neural Networks

Neetu Sharma and Keshav Dandeva

Abstract With the rapid evolution of deep convolutional neural networks (CNNs), major breakthroughs have been achieved in object detection in the field of computer vision. However, the majority of state-of-the-art detectors, in both one-stage and two-stage methods, have limits and are inadequate for usage in a real-world setting where each step must be thoroughly checked. This thesis investigates advanced object detection models and frameworks. It offers an in-depth analysis of the most recent object detection models, their frameworks, and the performance criteria used to evaluate such models. The object detection models selected are YOLOv5, Faster R-CNN using Detectron 2, and SSD using TensorFlow 2, and the dataset selected is the Vehicles-Open Images Dataset. The performance of the selected models in relation to many metrics is analyzed, and the findings are reported. In conclusion, the benefits and limits of the selected models, as well as their relative performance, are discussed.

Keywords Object detection · Computer vision · Deep convolutional neural network · CNN · YOLOv5 · Faster R-CNN · SSD · Detectron2 · TensorFlow 2

1 Introduction

Humans are capable of recognizing and locating objects in an image with the blink of an eye. The human visual system can carry out difficult tasks such as object identification and obstacle detection quickly and accurately and with minimal conscious effort. It would have been impossible to think that computers can do even half of that a couple of decades ago. Today however, due to the ease of access to enormous quantities of data, faster machines to process the data, and improved algorithms, computers

N. Sharma (✉)
Galgotias University, Greater Noida, Uttar Pradesh, India
e-mail: neetush75@gmail.com

K. Dandeva
Warsaw University, Warsaw, Poland

N. R. Shetty et al. (eds.), *Advances in Computing and Information*, Lecture Notes in Electrical Engineering 1104, https://doi.org/10.1007/978-981-99-7622-5_5

can be trained with almost no effort for detecting objects and labeling them for their respective classes in an image with high accuracy [1]. Since approximately 20 years, object detection has been evolving in working on detecting occurrences of visual objects belonging to a specific class in digital images. As a very demanding issue in computer vision, it is aiming to answer one question: "What objects are where?" This goal is trying to be accomplished by developing diverse computational models and techniques using state-of-the-art technologies like machine learning, artificial intelligence, and so on. Additionally, it constitutes the core foundation of numerous additional computer vision tasks including captioning of an image, instance segmentation, image captioning, tracking an object and-so-forth.

In addition to its importance in computer vision, object detection has a strong real-life importance nowadays. It is extensively applied in the real world in areas such as criminal investigation and autonomous driving (and robot vision). It plays an important, sometimes even crucial role in those domains as it reduces human involvement in certain very important processes therefore reducing errors. For instance, autonomous driving pursues to decrease the number of traffic accidents and optimizing traffic by reducing human involvement in the driving process. Going hand in hand with a crucial part of the autonomous driving technology is traffic sign and traffic light detection as well as pedestrian detection. It also improves life quality of people with different kinds of impairment, for example, text detection which allows visually impaired people to "read" street signs and currency.

Object detection may be less prone to error than humans, but it is still far from perfect. In each of the above-mentioned areas, object detection is facing some difficulties. The size of the pedestrians in the image or video can be too small, faces of people can be blocked partially or completely by other objects, texts can vary in fonts and languages and the image quality of traffic lights can be negatively impacted by bad weather conditions. These are just a few of many examples of challenges that can impair the accuracy of object detection. However, as mentioned, many types of object detection play crucial roles in real-world application and a wrong prediction can cause serious harm. Thus, in order to utilize the object detection model to its best capacity, it is vital to comprehend both its strengths and weaknesses.

With the advent of deep neural networks, a drastic improvement in the performance of object detection can be seen but still many of the methods are unable to measure how certain they are in their predictions. This becomes a considerable problem when a model is encountering previously unseen data as it may not be able to encode this input. For instance, if a model was trained with decent weather data and then faced with bad weather.

2 Methods

2.1 Dataset

In this, we analyze the class distribution and training/validation splits to define the dataset utilized for all experiments in this paper. The Vehicles-Open Images Dataset [2] supplied by Jacob Solawetz in Roboflow was used as the dataset for the implementation. The primary applications for this dataset are:

- Train an object detector to differentiate among a vehicle, bus, motorbike, ambulance, and truck
- Detector for objects at checkpoints for autonomous vehicle
- Examine the density of ambulances in automobiles with an object detector
- Train ambulance detector
- Explore the variety and precision of the Open Images dataset

The dataset contains a total of 627 photos of cars, with each image measuring 416 × 416 pixels. These photos contain numerous kinds of vehicles for object detection. These pictures are extracted from the Open Images open-source computer vision datasets. This collection represents only a fraction of the Open Images dataset for vehicles.

There are five designated categories, namely car, bus, motorcycle, truck, and ambulance. The dataset used is not proportionally balanced for all the classes, as is evident from the bar chart in Fig. 1. We can see that the class containing the label "car" is the one with the largest amount of instances in dataset, with nearly 651 labeled objects, followed by "bus" with 141, "motorcycle" with 140, "truck" with 136, and "ambulance" with 126. All other classes, excluding car, fall within the same range, which is considerably lower than the car class.

In this dataset, there are no missing annotations, meaning images that do not have an accompanying annotation file, or null examples, meaning images that deliberately do not contain any objects. In total, there are 1194 objects annotated, which roughly translates to 1.9 per image (average). The average image size, i.e., the size of images in megapixels, is 0.72 mp with images ranging from 0.29 mp to 1.05 mp.

For using the dataset in experiments conducted in the scope of this paper, the dataset was split in the ratio of 7:2:1 for the training dataset, validation dataset, and testing dataset, respectively. Resulting in 439 images being used as a training dataset, 125 images as a validation dataset, and 63 images as a testing dataset.

2.2 Object Detection

In this section, we will go through the chosen object detection models. For each of them, we will go through the research strategy used, all the components of the

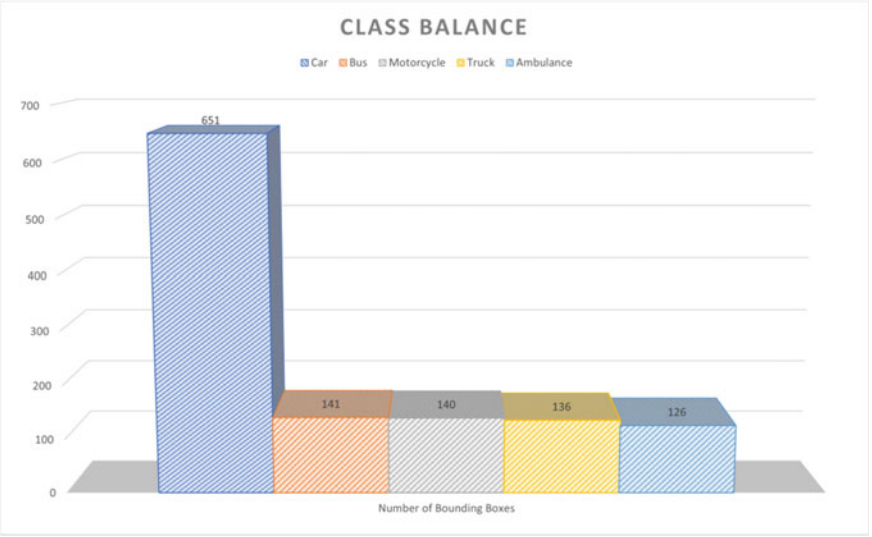

Fig. 1 Class distribution of the vehicles-open images dataset

model, the implementation of the model with the respective framework, the use of the dataset with respect to the model and the limitations faced.

All the object detection models are trained and tested using the online platform of Google Colab which provides free GPU support for running such tasks requiring great deal of computer resources.

2.3 YOLOv5

The first chosen object detection model for this paper is YOLOv5. The reason to go with YOLOv5 is fairly simple as it is the latest version of the YOLO object detection models with the advantage of being implemented in Python and having great accessibility for integration and use in most of the fields.

First, the repository of YOLOv5 provided by Ultralytics on GitHub is cloned. As this is the original source of the YOLOv5 implementation, any experiment based on YOLOv5 starts with the base of the code provided by Ultralytics for YOLOv5. Then, the requirements for running the model are installed. One of the main requirements, being, PyTorch as this is the framework on which YOLOv5 is built. Then, we also install the library for Roboflow as the dataset we are using is provided by Roboflow.

We must put up a dataset of sample images with bounding box annotations surrounding the things we wish to identify in order to train our own model. Additionally, we require a dataset in YOLOv5 format. As we are using Roboflow, we can convert our dataset for the YOLOv5 model using the Roboflow library.

3 Results

In this we will go through the results obtained from all the experiments performed. As mentioned before all the object detection models are used to train and test the same dataset, which is the Vehicles-Open Images Dataset [2]. The results are subdivided into the three subsections corresponding to the three object detection models chosen to perform the experiments. First, we will see the results from the individual object detection models, i.e., YOLOv5, Faster R-CNN, and SSD, and then we will compare them.

3.1 YOLOv5 Result

As mentioned in Sect. 3.2.1, the training process for the YOLOv5 model was completed in 27 min and 46 s. The model was supplied with the parameter for epochs set to 500, but it stopped the training process after 294 epochs as the model did not observe any improvement in the results in the last 100 epochs.

The brief summary of the YOLOv5 model that we used for this task can be expressed as 213 layers, 7,023,610 parameters, 0 gradients, and 15.8 GFLOPs. FLOP, or floating point operations per second, is a measure of performance, meaning how fast the computer can perform calculations. GFLOP is simply a Giga FLOP. We can see the metrics calculated for the validation dataset by the YOLOv5 model in Table 1.

Let us define the columns present in the table and their meaning.

- Class: This column enlists all the classes present in the dataset.
- Images: This column gives the number of images present in the validation dataset for which the respective classes were tested.
- Labels: This column depicts the total value of the respective class labels present in the validation dataset.
- Precision: This column is the measure of the precision, which is the number of correct bounding box predictions.
- Recall: This column is the measure of the recall, which is the number of true bounding boxes that were correctly predicted.

Table 1 YOLOv5 evaluation results

Class	Images	Labels	Precision	Recall	mAP_0.5	mAP_0.5:0.95
All	125	227	0.685	0.548	0.562	0.413
Ambulance	125	32	0.565	0.812	0.761	0.616
Bus	125	23	0.941	0.652	0.7	0.551
Car	125	119	0.653	0.403	0.474	0.321
Motorcycle	125	23	0.602	0.609	0.563	0.37
Truck	125	30	0.666	0.266	0.313	0.209

- mAP_0.5: This column is the measure of mean average precision (mAP) at an IoU (Intersection over Union) threshold of 0.5.
- mAP_0.5:0.95: This column is the measure of the average mAP over different IoU thresholds, ranging from 0.5 to 0.95.

From Table 1, we can see that for all the classes in the validation dataset, the precision is 68.5%, and the recall is 54.8%. The value of mAP_0.5 is 56.2% and mAP_0.5:0.95 is 41.3%. The highest value of mAP_0.5 is obtained for the "Ambulance" class, which is 76.1%, whereas the lowest is for the "Truck" class, which is, 31.3%. The trend stays the same for the mAP_0.5:0.95 values, but with an overall decrease in the measures.

We can see the trend of the above mentioned metrics (precision, recall, mAP_0.5, and mAP_0.5:0.95) in the graphs presented in Figs. 2, 3, 4 and 5. The graphs are generated with a smoothing value of 0.6 for the clarity of the depiction of trends.

For better understanding of the results, we should also look at the losses in the YOLOv5 model. Yolo's loss function is composed of three parts. One, box_loss which is the bounding box regression loss (also called Mean Squared Error). Second, object loss which is the confidence score of presence of object. It is the objectness loss (also called Binary Cross Entropy). Third, class loss which is the classification loss. As we can see from the figures above the box_loss is reduced to 0.02 by the last step of training, in contrast to it being started from a value of 0.11 in the first step. The same trend is observed for obj_loss and cls_loss. The cls_loss value started at 0.05 in the first step and was reduced to 1.31e-3 in the last step, and the obj_loss value started at 0.04, and by the last step it was reduced to 0.02.

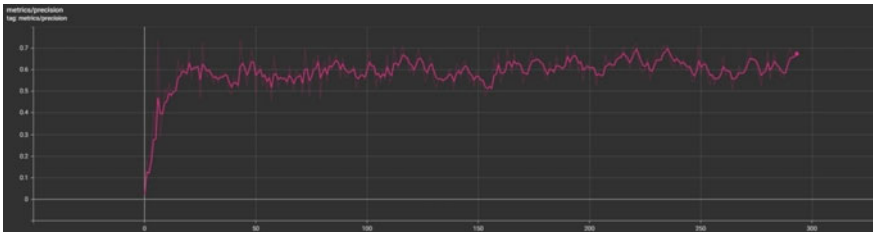

Fig. 2 Precision versus epochs

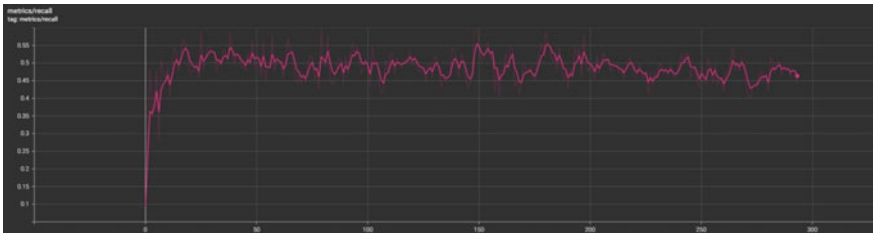

Fig. 3 Recall versus epochs

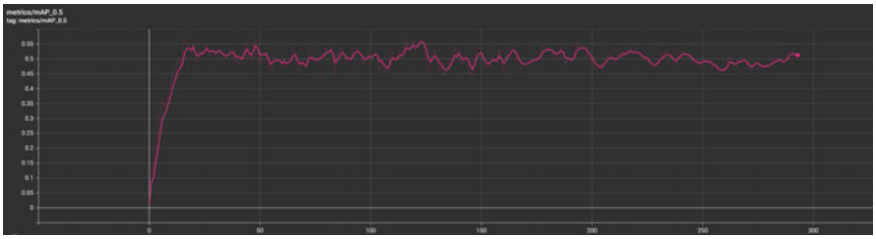

Fig. 4 mAP_0.5 versus epochs

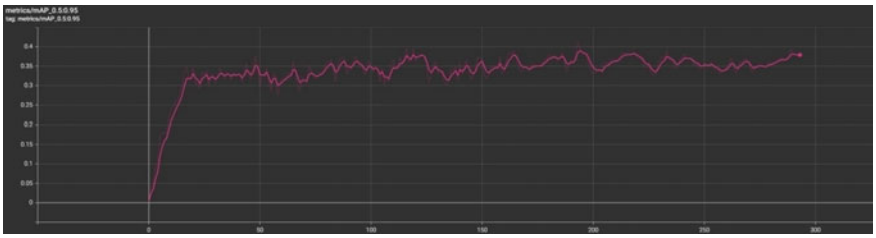

Fig. 5 map_0.5–0.95 versus epochs

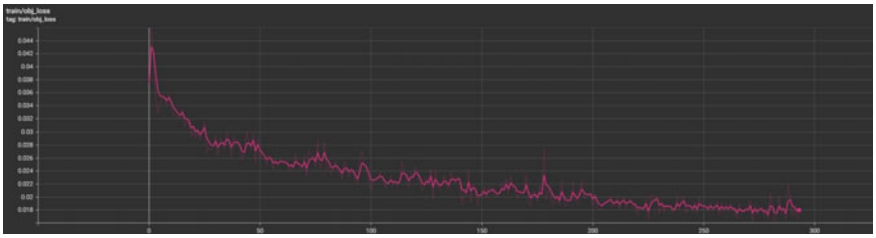

Fig. 6 obj_loss value versus epochs

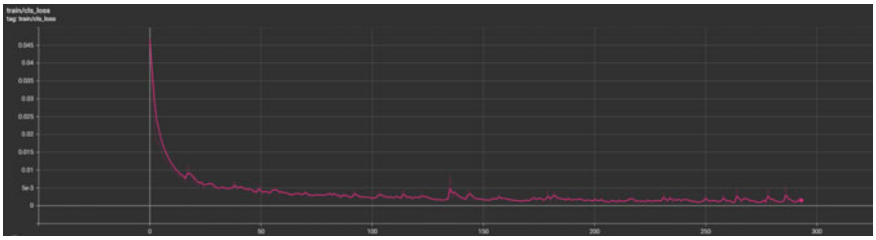

Fig. 7 cls_loss value versus epochs

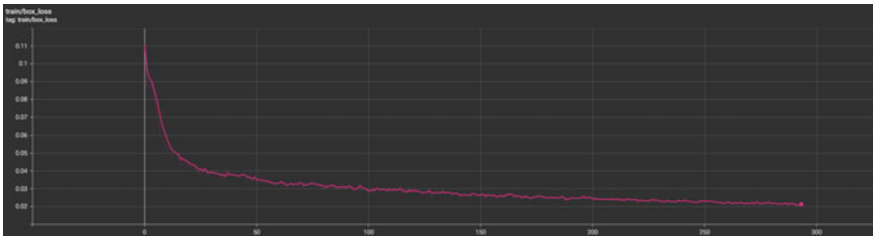

Fig. 8 box_loss value versus epochs

Finally, after training the model and obtaining satisfying performance results, we run inference with the trained weights on the test dataset. The final outcomes after running the inferences are saved, and the images with the bounding boxes and predictions are displayed. The speed of the model for running inference on the test dataset is 0.5 ms preprocess, 13.7 ms inference, and 1.7 ms NMS (non-maximum suppression) per image at shape (1, 3, 416, 416).

3.2 Faster R-CNN Result

As mentioned earlier after setting the parameters and starting the training process of the Faster R-CNN model, it completed the process in 10 min and 29 s for 2000 iterations. As per the overall training speed for the 2000 iterations, it took approximately 0.31 s per single iteration. For training, the model loaded 439 images in the COCO format from the given location of the training dataset. There were no images found with any usable annotations. The distribution of the instances among all the categories is presented in Table 2.

During the training process, we can observe various performance metrics and loss function evaluations for the model. In Fig. 9, we can see that during the early steps of the process, the false negative value peaked at 0.98, but over the course of the process and by the last iteration, i.e., at step 2000, the value is reduced to approximately 0.05. In Fig. 10, we can observe that the class accuracy during the training process started from a value of 0.3 in the beginning steps but soon climbed up to 0.93 in 300 steps

Table 2 Distribution of instances for training Faster R-CNN model

Category	Instances
Ambulance	85
Bus	99
Car	457
Motorcycle	101
Truck	96
Total	838

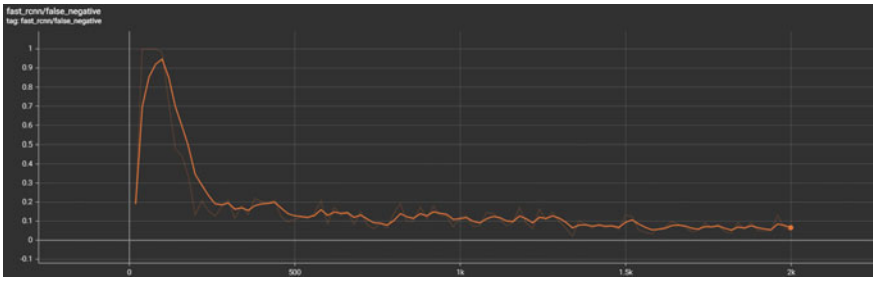

Fig. 9 False_negative value versus steps

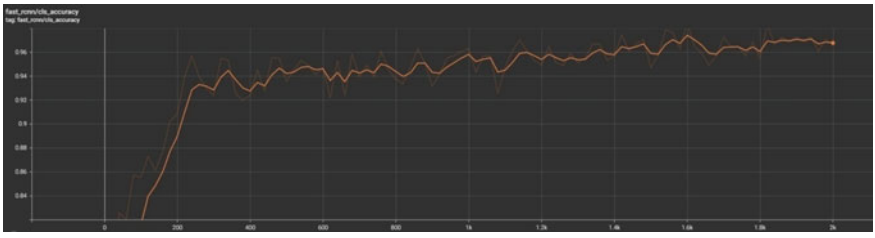

Fig. 10 cls_accuracy value versus steps

and then maintained an almost steady value with the peak value being 0.98 near the final iterations.

One of the important metrics to note during the training process are the loss functions. As we can see in Fig. 11, the value of the loss_box_reg starts at more than 0.6 in the first 80 steps but then reduces to less than 0.3 after 200 steps, and finally is the lowest toward the final steps with the value of 0.17. The "loss_box_reg" is localization loss present in the Region of Interest head which calculates the loss for localization of box. From the Fig. 12, we can see the value of loss_cls being reduced from 1.73 in the first iteration to 0.08 in the last iteration. "loss_cls" is the classification loss in the Region of Interest head which calculates the box classification loss, i.e., the value of the performance of model in labeling a correct class. In Fig. 13, we can see that the value of the "loss_rpn_cls" starts from 0.022 in the beginning and then fluctuates between $9.81e-3$ and $2.75e-3$ for iterations after 400 and before 1400. Finally, at the end of the training process, the value reduces to $2.99e-3$. This value corresponds to classification loss in RPN which calculates the "objectness" loss, i.e., the value of the performance of model in labeling the foreground anchor boxes and background anchor boxes correctly. Finally, we take a look at the total loss which is the weighted sum of all the individual losses calculated during the iterations, in the Fig. 14. We can see that total loss follows the same trend of starting at a high value of 2.34 in the first steps and then finally reducing to value of 0.25 by the last step.

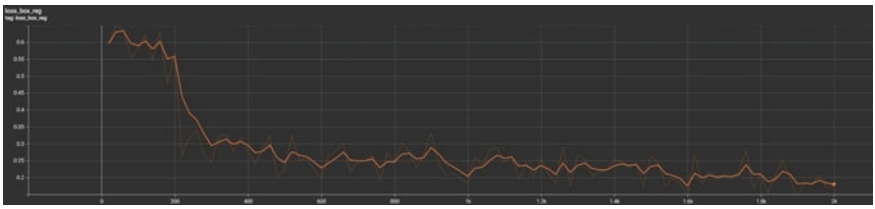

Fig. 11 loss_box_reg value versus steps

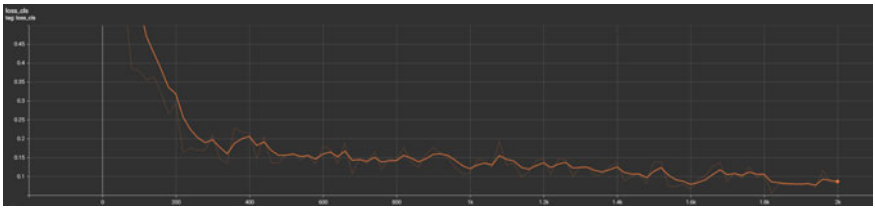

Fig. 12 loss_cls value versus steps

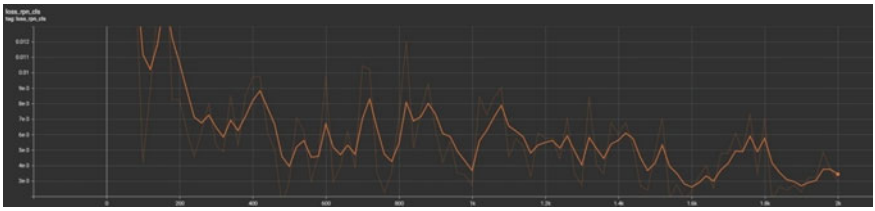

Fig. 13 loss_rpn_class value versus steps

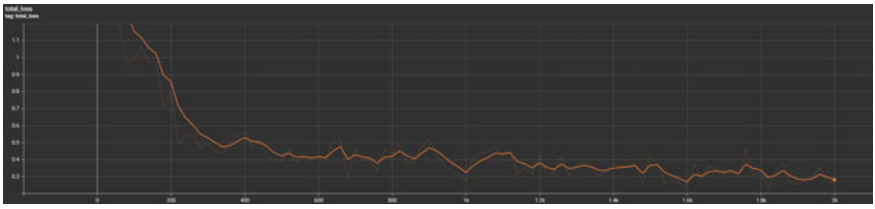

Fig. 14 Total_loss value versus steps

After the training process is complete with satisfying metrics, we run the inference on the model using the test dataset. The time taken by the model for testing the dataset was 9 min and 57 s, with an average of 0.165 s per iteration. For testing, the model loaded 63 images in the COCO format from the given location of the testing dataset. The distribution of the instances among all the categories is presented in Table 3.

Table 3 Distribution of instances for testing Faster R-CNN model

Category	Instances
Ambulance	9
Bus	19
Car	75
Motorcycle	16
Truck	10
Total	129

The evaluations from the inference on the testing dataset are presented in Tables 4 and 5.

Table 4 Evaluation of the Faster R-CNN model

AP for IoU = 0.50:0.95	Area = all	Maximum detections = 100	= 0.400
AP for IoU = 0.50	Area = all	Maximum detections = 100	= 0.613
AP for IoU = 0.75	Area = all	Maximum detections = 100	= 0.440
AP for IoU = 0.50:0.95	Area = small	Maximum detections = 100	= 0.126
AP for IoU = 0.50:0.95	Area = medium	Maximum detections = 100	= 0.104
AP for IoU = 0.50:0.95	Area = large	Maximum detections = 100	= 0.505
AR for IoU = 0.50:0.95	Area = all	Maximum detections = 1	= 0.363
AR for IoU = 0.50:0.95	Area = all	Maximum detections = 10	= 0.477
AR for IoU = 0.50:0.95	Area = all	Maximum detections = 100	= 0.478
AR for IoU = 0.50:0.95	Area = small	Maximum detections = 100	= 0.277
AR for IoU = 0.50:0.95	Area = medium	Maximum detections = 100	= 0.149
AR for IoU = 0.50:0.95	Area = large	Maximum detections = 100	= 0.580

Table 5 Evaluation results per category for bbox of the Faster R-CNN model

Category	AP
Car	38.262
Ambulance	78.344
Motorcycle	33.365
Bus	49.999
Truck	0.000

3.3 SSD Result

As mentioned in earlier, we configured the pipeline.config file with parameters for the training process. Before starting the actual model training, it took a few minutes to set up everything and read the provided pipeline file and training.record file generated for the dataset. The actual training process took 46 min and 31.56 s for 5000 steps. During the training process, we can observe various performance metrics and loss function evaluations for the model. For every step, we can observe the values for these functions. One of the main metrics for the training process is the learning rate of the model. We used the default values for the pretrained model for setting the base learning rate and its related values. From Fig. 15, we can see that the learning rate during the training process started with a steady increase from 0 to up until 0.4 in the first 2500 steps, then it maintained the learning rate around that value for the next 1500 steps, but after that had a gradual dip and ended up at 0.38 by the final step of the training.

Another important metric to keep note of during the training process is the loss function. The loss function, in general, is the measure of how successfully the model can assess a certain amount of information in a given algorithm based on the quantity of data provided. For this model, during training there were logs of four different loss functions. First one being, classification loss which loosely means the classes or the labels correctly identified are more than the incorrectly identified ones. From the Fig. 16, we can see that the classification loss value started at a high point of 0.89 in the beginning of the training process but then steadily decreased throughout with exception of a bump around 1500 steps and finally reducing to the lowest value of 0.32 after 5000 steps.

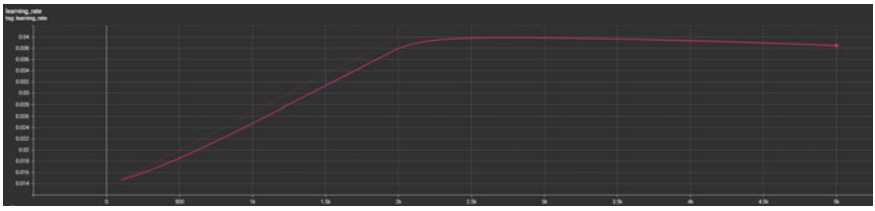

Fig. 15 Learning rate versus steps

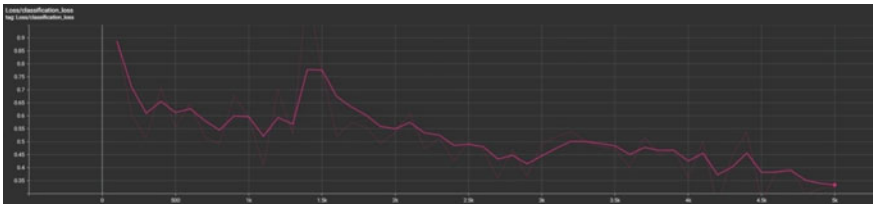

Fig. 16 Classification loss versus steps

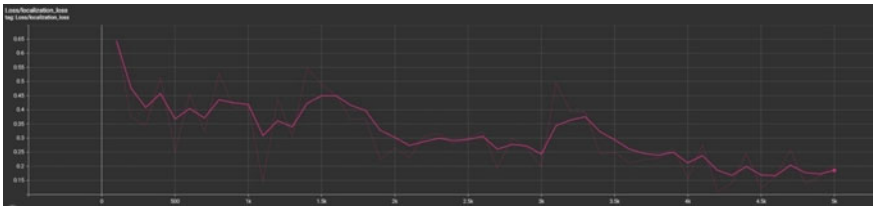

Fig. 17 Localization loss versus steps

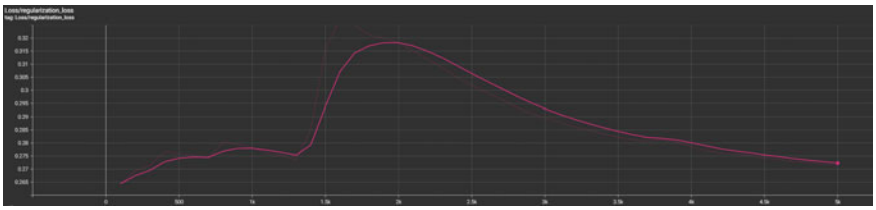

Fig. 18 Regularization loss versus steps

A second loss function that we have for this training process is the localization loss, which is a smooth L1 loss between the projected bounding box adjustment and the actual values. From Fig. 17, we can see that the value for this loss started at 0.64 in the initial steps of training and then had a steady decline with a bit of fluctuation in the values but overall reducing to 0.2 by the end of the training process.

A third loss function that is noted during this training process is regularization loss. Regularization process helps the model generalize better. The regularization loss is the loss generated by the regularization functions. For our training process, as we can see in Fig. 18, the value of the regularization loss started at a low of 0.26 in the initial steps, then had a steep increase after 1400 steps and had its peak value of 0.32 around 2000 steps, but then had a steady decline and finally reached the value of 0.27 by the end of the training process, i.e., at 5000 steps.

Finally, we have the total loss function, which is the weighted sum of all the individual losses calculated during the steps of the training process. From Fig. 19, we can observe that the value of the total loss function started at 1.8 in the initial steps of training but then had a great decrease in the value and reached around 1.3 by step 300. After that, the value had a steady decline but with various fluctuations along the way, with a peak value of 1.58 around 1500 steps and then the lowest value at the very last step of the training, 0.8.

After the successful completion of the training process, now we start with the testing process. For testing the model, 63 images were provided as per the dataset split, and the model performed the testing and evaluation in less than 1 min. The results from the evaluation are present in Tables 6 and 7 for the final step.

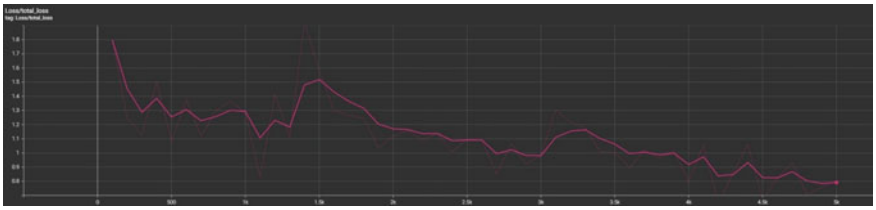

Fig. 19 Total loss versus steps

Table 6 Evaluation results for the SSD model

AP for IoU = 0.50:0.95	Area = all	maximum detections = 100	= 0.222
AP for IoU = 0.50	Area = all	Maximum detections = 100	= 0.357
AP for IoU = 0.75	Area = all	Maximum detections = 100	= 0.264
AP for IoU = 0.50:0.95	Area = small	Maximum detections = 100	= 0.087
AP for IoU = 0.50:0.95	Area = medium	Maximum detections = 100	= 0.048
AP for IoU = 0.50:0.95	Area = large	Maximum detections = 100	= 0.276
AR for IoU = 0.50:0.95	Area = all	Maximum detections = 1	= 0.283
AR for IoU = 0.50:0.95	Area = all	Maximum detections = 10	= 0.395
AR for IoU = 0.50:0.95	Area = all	Maximum detections = 100	= 0.409
AR for IoU = 0.50:0.95	Area = small	Maximum detections = 100	= 0.171
AR for IoU = 0.50:0.95	Area = medium	Maximum detections = 100	= 0.254
AR for IoU = 0.50:0.95	Area = large	Maximum detections = 100	= 0.483

Table 7 Evaluation of loss functions for the SSD model

Localization loss	0.324438
Classification Loss	0.566739
Regularization Loss	0.271516
Total Loss	1.162693

3.4 Evaluation

In this section, we will compare the results obtained from the chosen object detection models for this paper, i.e., YOLOv5, Faster R-CNN using Detectron2, and SSD using TensorFlow 2. The models are trained and tested on the same dataset in order to achieve results that can be compared quantitatively. We will evaluate the performance of the before mentioned models on some common performance metrics and also take a look at the time taken by each model to train on the dataset to acquire the presented results.

Table 8 Evaluation of loss functions for the SSD model

Model	Total training time	Total iterations	Average time/step (s)
YOLOv5	27 min 46 s	11,760	0.142
Faster R-CNN	10 min 29 s	2000	0.315
SSD	46 min 31.56 s	5000	0.558

3.5 Training Time

The time the models required for training on the dataset is presented in the Table 8. As we can see, the YOLOv5 model completed the training in 27 min and 46 s for 294 epochs, which results in 11,760 iterations, whereas the Faster R-CNN model took 10 min and 29 s for 2000 iterations and the SSD model completed the training process in 46 min and 31.56 s for 5000 iterations. This gives us the average time taken by the models per step, as presented in the Table 8. Although Faster R-CNN took the least amount of time for training overall, YOLOv5 has significantly less time per step. The time taken by YOLOv5 model per iteration is almost half of what the Faster R-CNN takes. The SSD model has the longest time per iteration for the training process.

3.6 Performance Metric

One of the most important metrics to measure the performance of any object detection model is mean Average Precision (mAP). The value of mAP is the score generated by comparing the bounding box of ground truth with the bounding box detected by the model. The higher the value obtained for this metric mAP, the more precise are the findings of the model. mAP@0.5 corresponds to the mAP for Intersection over Union (IoU) thresholds 0.5, and mAP@0.5:0.95 corresponds to the average mAP for IoU thresholds from 0.5 to 0.95 with a step size of 0.05 (0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95). We can see the evaluation results from the three models for the mAP@0.5 values and mAP@0.5:0.95 values for the test dataset in Table 9.

As is evident from the Table 9, for mAP with IoU threshold 0.5, the best results are obtained from the model Faster R-CNN with the precision percentage being equal to 61.3%. YOLOv5 comes in as close second with 56.2% for the same metric and SSD comes in last with a low value of 35.7%. The map@0.5 value for Faster R-CNN

Table 9 Comparison of mAP@0.5 and mAP@0.5:0.95

Model	mAP@0.5	mAP@0.5:0.95
YOLOv5	0.562	0.413
Faster R-CNN	0.613	0.400
SSD	0.357	0.222

is almost double than of SSD with YOLOv5 being only around 8% less than Faster R-CNN's value.

For the metric of average mAP calculated over IoU thresholds ranging from 0.5 to 0.95 with a 0.05 step value, we can see that YOLOv5 takes the first stop with the highest value among all the three models of 41.3%. Faster R-CNN comes a close second with a value of 40%, which is only slightly less than YOLOv5. The SSD model scores the lowest value again in this metric as well, with a percentage of 22.2%. The mAP@0.5:0.95 value for SSD is almost half of the values obtained by the other two models.

4 Conclusion

The work done under the scope of this paper constitutes state-of-the-art object detection models and deep learning neural network frameworks. The purpose of the present degree project is to research the latest object detection models in deep convolutional neural networks, examine prominent deep object detection frameworks, and evaluate their performance for a certain dataset.

After researching about the object detection models and a brief literature review of the object detection algorithms and frameworks, three object detection models, namely, YOLOv5 (You Only Look Once), Faster R-CNN (Regional Convolutional Neural Network) using Detectron2 framework, and SSD (Single Shot Detector) using TensorFlow 2 framework, were selected to perform experiments and then performance evaluation. The dataset was chosen with images related to autonomous vehicle driving, and the size of the dataset chosen was relatively small due to some delimitation and to allow the experiments to be conducted in a reasonable amount of time.

All the models were trained and tested using Google Colab online environment. The YOLOv5 model was implemented using the online available source code from the authors at Ultralytics GitHub, the Faster R-CNN model was implemented using the Detectron2 framework source code available for various pretrained model in the official GitHub repository of Facebook AI Research Team, and the SSD model was implemented using the TensorFlow 2 framework's official GitHub repository with pretrained models. All the models were trained on the selected custom dataset and optimized as required.

The models YOLOv5 and Faster R-CNN outperformed the SSD model by a great margin in terms of time taken for training and the final mAP value for the test dataset. The YOLOv5 model proved to be the fastest, taking less than half the time taken by the Faster R-CNN model in training. Faster R-CNN model proved to be slightly more accurate than YOLOv5 for mAP@0.5, but for mAP@0.5:0.95 metric, YOLOv5 outperformed Faster R-CNN model by a slight margin. Concluding, the findings of the thesis are in clear favor of YOLOv5 in terms of speed and accuracy, but with a slight advantage to Faster R-CNN in terms of accuracy for a trade-off of speed.

References

1. Miya J, Ansari MA (2020) Medical images performance analysis and observations with spiht and wavelet techniques. J Inf Optim Sci 41:273–282. https://doi.org/10.1080/02522667.2020.1721616
2. Solawetz J (2020) Vehicles-openimages object detection dataset. Roboflow. [Online]. Available: https://public.roboflow.com/object-detection/vehicles-openimages
3. Girshick R, Donahue J, Darrell T, Malik J (2014) Rich feature hierarchies for accurate object detection and semantic segmentation. In: 2014 IEEE conference on computer vision and pattern recognition. https://doi.org/10.1109/cvpr.2014.81. [Online]. Available: https://dl.acm.org/citation.cfm?id=2679851
4. Girshick R (2015) Fast r-cnn. In: 2015 IEEE international conference on computer vision (ICCV), pp 1440–1448. https://doi.org/10.1109/iccv.2015.169
5. Ren S, He K, Girshick R, Sun J (2017) Faster R-CNN: towards real-time object detection with region proposal networks. IEEE Trans Pattern Anal Mach Intell 39:1137–1149. https://doi.org/10.1109/tpami.2016.2577031
6. Redmon J, Divvala S, Girshick R, Farhadi A (2016) You only look once: unified, real-time object detection. In: 2016 IEEE conference on computer vision and pattern recognition (CVPR). https://doi.org/10.1109/cvpr.2016.91. [Online]. Available: https://arxiv.org/pdf/1506.02640.pdf
7. Redmon J, Farhadi A (2017) Yolo9000: better, faster, stronger. In: 2017 IEEE conference on computer vision and pattern recognition (CVPR). https://doi.org/10.1109/cvpr.017.690
8. Redmon J, Farhadi A (2018) Yolov3: an incremental improvement. arXiv preprint arXiv:1804.02767. [Online]. Available: https://arxiv.org/abs/1804.02767
9. Liu W, Anguelov D, Erhan D et al (2016) Ssd: Single shot multibox detector. In: Computer vision—ECCV 2016, pp 21–37. https://doi.org/10.1007/978-3-319-46448-0_2. [Online]. Available: https://arxiv.org/abs/1512.02325
10. Yu F, Chen H, Wang X et al (2020) Bdd100k: a diverse driving dataset for heterogeneous multitask learning, 2020. [Online]. Available: https://openaccess.thecvf.com/content_CVPR_2020/papers/Yu_BDD100K_A_Diverse_Driving_Dataset_for_Heterogeneous_Multitask_Learning_CVPR_2020_paper.pdf
11. Lomonaco D, Maltoni (2015) Deep learning for computer vision: a comparison between convolutional neural networks and hierarchical temporal memories on object recognition tasks. [Online]. Available: https://amslaurea.unibo.it/9095/1/Vincenzo_Lomonaco_tesi.pdf
12. Contributors W (2018) Artificial neural network, Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Artificial_neural_network
13. Emmons J, Fouladi S, Ananthanarayanan G, Venkataraman S, Savarese S, Winstein K (2019) Cracking open the DNN black-box. In: Proceedings of the 2019 workshop on hot topics in video analytics and intelligent edges—HotEdgeVideo. https://doi.org/10.1145/3349614.3356023
14. What are neural networks? IBM, Aug. 2020. [Online]. Available: https://www.ibm.com/cloud/learn/neural-networks
15. Camacho C (2018) Convolutional neural networks. [Online]. Available: https://cezannec.github.io/Convolutional_Neural_Networks/
16. Stoycheva M (2021) Uncertainty estimation in deep neural object detectors for autonomous driving
17. Brownlee J (2019) A gentle introduction to object recognition with deep learning, Machine Learning Mastery. [Online]. Available: https://machinelearningmastery.com/object-recognition-with-deep-learning
18. Krizhevsky A, Sutskever I, Hinton GE (2012) Imagenet classification with deep convolutional neural networks. Commun ACM 60:84–90. https://doi.org/10.1145/3065386
19. Padilla R, Passos WL, Dias TLB, Netto SL, da Silva EAB (2021) A comparative analysis of object detection metrics with a companion open-source toolkit. Elec tronics 10. https://doi.org/10.3390/electronics10030279. [Online]. Available: https://www.mdpi.com/2079-9292/10/3/279

20. Everingham M, Van Gool L, Williams CKI, Winn J, Zisserman A (2009) The pascal visual object classes (voc) challenge. Int J Comput Vision 88:303–338. https://doi.org/10.1007/s11263-009-0275-4
21. Lin TY, Maire M, Belongie S et al (2014) Microsoft coco: common objects in context. Computer vision—ECCV 2014, pp 740–755, 2014. https://doi.org/10.1007/978-3-319-10602-1_48. [Online]. Available: https://link.springer.com/chapter/https://doi.org/10.1007/978-3-319-10602-1_48
22. Deep learning frameworks, NVIDIA Developer, Apr. 2016. [Online]. Available: https://developer.nvidia.com/deep-learning-frameworks
23. Bradski G (2000) The OpenCV Library. Dr. Dobb's J Softw Tools
24. TensorFlow, *Why tensorflow*, TensorFlow. [Online]. Available: https://www.tensorflow.org/about
25. *Detectron2*, Facebook. [Online]. Available: https://ai.facebook.com/tools/detectron2/
26. Keras, *Keras documentation*, Keras, 2019. [Online]. Available: https://keras.io/
27. Russakovsky O, Deng J, Su H et al (2015) Imagenet large scale visual recognition challenge. Int J Comput Vis 115:211–252. https://doi.org/10.1007/s11263-015-0816-y. [Online]. Available: https://hci.stanford.edu/publications/2015/scenegraphs/imagenet-challenge.pdf