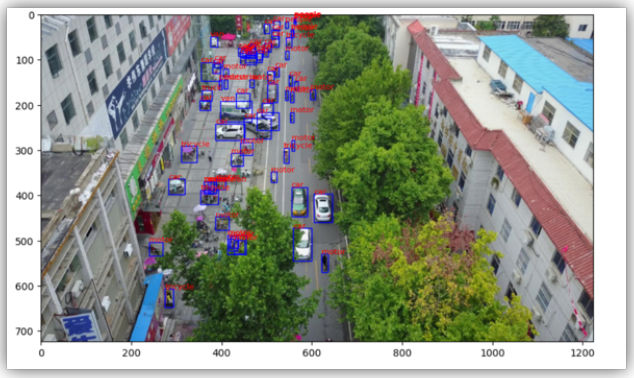
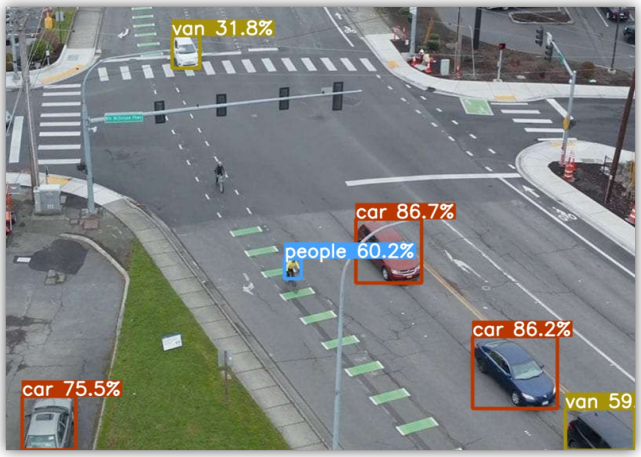
Comparative Assessment of Advanced Deep Learning & Computer Vision Models for Real-Time Aerial Object Detection in Drone Surveillance

Nimesh Arora, Dikshant Sagar, Ishmael Valenzuela, Ashish Gurung, Safal Rijal, Kavya Reddy, Shreyas Teli, Harika Florence

{nimesh939arora, sagar.dikshant, ismaelrvalenzuel4, ashishgurung1996, itssafal111, kavyareddy.b333, telishreyas10, harikaflorence23}@gmail.com





Abstract

In this project, we aim to accurately detect vehicles in drone-captured images using various computer vision models. To accomplish this, we utilize the VisDrone2019 dataset, which consists of annotated images and videos captured by drones in different locations, environments, and weather conditions. We evaluate the performance of several popular models, including RCNN, Faster RCNN, Mask RCNN, Single Shot Detector (SSD), Yolo 3, Yolo 5, and Yolo 8, and compare their results. Our findings provide insights into the effectiveness of different models for vehicle detection in drone imagery.

1. Introduction/Background/Motivation

The objective of our project is to accurately detect and identify vehicles in aerial footage captured by drones. This can be a challenging task due to the varying camera angles, lighting conditions, and complex backgrounds. We aim to use advanced computer vision techniques like deep learning models to tackle this problem and provide accurate and efficient results. [10]Our goal is to enable better analysis and understanding of aerial footage for a wide range of applications, such as traffic monitoring, surveillance, and urban planning.

Currently, object detection is done using deep learning models trained on annotated images. These models extract

features from the images and then classify and localize the objects present in the image. However, the accuracy of these models heavily relies on the quality of the annotations provided in the training data. [9] This means that errors in the annotations can lead to inaccurate object detection results. Furthermore, these models require large amounts of annotated data to achieve high accuracy, which can be time-consuming and expensive to obtain.

The limits of current object detection practice include the challenges of obtaining high-quality annotations, which can be subjective and prone to human error. [8]Additionally, the high computational requirements of deep learning models can make them impractical for real-time applications or devices with limited processing power. Furthermore, some models may struggle with detecting objects that are heavily occluded or have complex shapes, leading to inaccurate detections. Overall, while current object detection methods have made significant progress, there is still room for improvement in terms of accuracy, efficiency, and robustness.

If our effort is successful, it has the potential to have a significant impact on the field of computer vision. Our work can assist to automate the process of object recognition and tracking in aerial photos and videos, which can be utilized for surveillance, search and rescue, and traffic monitoring, among other things. Our work can help save lives, avoid accidents, and increase safety in many contexts by enhancing the precision and efficiency of these jobs. Our study may potentially open the way for more advanced and sophisticated computer vision technologies to be used in fields such as robotics, agriculture, and healthcare.

We used the VisDrone dataset to train our model. The dataset contains thousands of images and videos of different environments captured by drones. The dataset is labeled with various annotations such as object bounding boxes, object categories, occlusion levels, and more. The dataset is publicly available and can be accessed through this link: https://paperswithcode.com/dataset/visdrone.

The VisDrone dataset contains object bounding boxes, which are the outlines that surround the objects in the images and videos. It also includes object categories, which tell us what type of object is in the bounding box, like a car or a person. The dataset also has occlusion levels and truncation levels that show how much of the object is hidden. All these labels help us teach our computer vision models to detect, track, and count objects accurately.

2. Models

To tackle the problem of object detection in photos and videos, we applied several computer vision techniques. To recognize objects in photos and videos, we used algorithms such as Mask R-CNN, SSD, RCNN, Faster R-CNN, YOLO v3, v5, and v8.

We assumed that applying these methods would be beneficial in tackling the problem of object detection because prior research investigations had shown promising outcomes. We took the approach of implementing these algorithms and testing their performance on our dataset.

In our approach, we tested many algorithms to discover the optimal one that met our requirements. We also employed transfer learning techniques to increase the accuracy of our results by fine-tuning pre-trained models on our dataset.

Our method is not wholly novel, but we did use cutting-edge object identification algorithms and approaches. Additionally, we fine-tuned the models with our dataset, which is unique to our issue area.

2.1. Single Shot Detector (SSD)

[1] SSD is a fully convolutional network for object detection that predicts bounding boxes and class scores directly from image pixels. It divides the image into a grid of cells and predicts multiple bounding boxes and class scores for each cell at different scales and aspect ratios. The network is trained end-to-end on a joint loss function that combines the localization and classification losses.

2.2. RCNN

Region-based Convolutional Neural Network (R-CNN) is a method of object detection in images.it first offers candidate bounding boxes in an image using a method such as selective search then extracts from each proposed region a (CNN) Convolutional Neural Network. The features are then put into a classifier to find out the class of the object. A regressor then generates and refines the various bounding box coordinates.

2.3. Faster R-CNN

[7]Faster R-CNN is an extension of R-CNN that replaces the selective search object proposal method with a Region Proposal Network (RPN) that predicts objectness scores and bounding box offsets at each location in a feature map. The RPN generates a set of object proposals that are then classified using a separate CNN. The two networks are trained jointly end-to-end on a joint loss function that combines the RPN and classification losses.

2.4. Mask R-CNN

[7]Mask R-CNN is an extension of Faster R-CNN that adds a segmentation mask prediction branch on top of the existing object detection branch. It uses the same Region Proposal Network (RPN) to generate proposals but adds an extra branch to predict object masks for each proposal. The network is trained end-to-end on a joint loss function that combines object detection and mask segmentation losses.

2.5. YOLO v3

[6] YOLO (You Only Look Once) is a one-stage object detection framework that divides the image into a grid of cells and predicts bounding boxes and class scores directly from each cell.[5] YOLO v3 uses a feature extraction network based on DarkNet-53 and multiple prediction scales to improve the accuracy of small object detection. The network is trained end-to-end on a joint loss function that combines the localization and classification losses.

2.6. YOLOv5

[3] YOLOv5 is an upgraded version of YOLO that achieves state-of-the-art object detection performance on a number of benchmark datasets. [2]It uses a novel architecture with a focus on speed and simplicity, achieving high accuracy with fewer parameters than previous versions of YOLO. YOLOv5 also includes a number of optimization techniques such as mixed-precision training and auto-mixed-precision.

2.7. YOLOv8

[4] YOLOv8 is a recent advancement in the YOLO family of object detection algorithms that uses a hybrid approach combining the strengths of both YOLOv3 and YOLOv5. [5] It achieves state-of-the-art performance on several object detection benchmarks while maintaining real-time inference speed. YOLOv8 also incorporates a number of design improvements such as a novel loss function and feature pyramid network.

3. Approach

The images were imported from the file for preprocessing using the Python Imaging Library. They were then downsized to a predetermined size to guarantee that all of the photographs had the same proportions, which is essential for neural network input. The photos were then transformed into PyTorch tensors, which are multidimensional arrays that can be fed into the neural network. The picture pixel values were then normalized by removing the mean pixel value and dividing it by the standard deviation of all photos in the dataset. This is done to guarantee that the neural network's input has a zero mean and unit variance, which can assist enhance the model's performance. The pictures are then transformed to the Faster R-CNN model's necessary format by transposing the tensor and transforming the color channel from RGB to BGR format. These processes guarantee that the photos are correctly structured and ready to be fed into the object detection model.

3.1. Single Shot Detector (SSD)

We utilized Single Shot MultiBox Detector design has a special variant called SSDLite320MobileNetV3Large that makes use of the MobileNetV3 Large network as its backbone over the vis-drone dataset. In order to construct a model that could precisely identify different things in aerial photos from the VIsdrone 2019 dataset, We used the SSD PyTorch model as part of my object detection project. I carried out the following actions to accomplish this: We received the VIsdrone 2019 dataset and enhanced the remaining photographs to make the dataset larger after deleting any corrupted ones. The information was also divided into training and validation sets. Using PyTorch, We trained the Mask RCNN model using 20 iterations, a 32-batch size, and a learning rate of 0.005. With an SSDLite320MobileNetV3Large pre-trained model, We additionally performed transfer learning to hone the model using the VIsdrone dataset. Using metrics for mean average precision (mAP) and intersection over union (IoU), We assessed the trained model's performance. The model's mAP and IoU of 0.5 show that it was successful in accurately identifying items in the VIsdrone photos. However, We also had to deal with issues like class imbalance and overfitting throughout the evaluation process. Overall, the project's findings show how well the SSDLite320MobileNetV3Large model performs object detection in aerial photos and illustrates the potential uses for this technology in surveillance and remote sensing. Future research may be able to use more sophisticated methods, including multi-task learning or attention mechanisms, to enhance the model's performance. During the training of the SSD model, we faced a few challenges. One of the main problems was related to the training process. As we had a limited amount of VRAM, which is required to train large datasets, we had to face issues such as the model not continuing during the training process. Despite trying a few solutions, the first thing we tried did not work effectively. As with any object detection model, there were concerns about how well the Single Shot Detector (SSD) model would perform on the VisDrone dataset, especially given the variability of drone footage. There was also some concern that SSD might struggle with smaller or more distant objects due to its use of a single-shot approach, rather than using region proposals like Faster R-CNN. Additionally, since SSD is not one of the most recent object detection models, there was some uncertainty about its effectiveness in comparison to newer models. We were successful in training the SSD model to detect objects in the VisDrone dataset accurately. The model was able to identify the objects for which it was trained with a satisfactory level of precision and could detect objects in real time with high accuracy.

3.2. RCNN with Resnet50-FPN and Selective Search Algorithm

The approach initially involved the pre-processing of the VisDrone2019 dataset which is a collection of the drone-

based images that come with annotation of each of the images as a .txt format. Each of these images together with their annotations were loaded into the system forming the pillar of the training data. To ensure a uniform input size and the model compatibility the images were reduced to a resolution of 480 x 480 pixels this process was done without interfering with the critical visual information in all the given images. This step was considered very important because it reduced the computational requirements given the limited GPU available in the google collab infrastructure. To generate the region proposals there was need to introduce the selective search algorithm which operates by grouping together similar pixels based on the shape compatibility, texture, size, and colour .and impose it on the Faster RCNN model already exposed and pretrained by Torchvision. This effectively captures objects of different scales present in the image because adding the selective search layer causes it to have multiple region proposals of varying scales. During the training it is important to note that the learning rate determines the step size at each iteration while moving towards a minimum of a loss function. In this approach the learning rate was dynamically managed and after each epoch the learning rate was systematically lowered according to the schedule that was already predetermined. This step is known as learning rate annealing which ensures the models convergence to a global minimum thereby improving the overall performance and stability of the model. Adjusting the learning rate is very key as it prevents over fitting ensuring the models robustness and its generalizability to data unseen. The model was then assessed using the VisDrone2019-val dataset which similarly to VisDrone2019-train dataset is comprised of the various images in .jpg format and .txt annotation files that is individualized for each of the images in the dataset. The ground truth bounding boxes were then used as a foundation of this assessment the predicted bounding boxes were then tested against these ground truths. This assessment procedure also incorporated a non-maximum suppression (NMS) phase in order to eliminate the overlapping and redundant bounding boxes and maintain the most confident ones. The confidence scores were then calculated as a percentage written on the top right of the bounding box with the object name of the top left of the bounding box. Through out this approach very careful steps were taken at each stage to ensure that the optimization of the detection performance is as efficient and effective as possible using the faster RCNN architecture together with the selective search algorithm through the dynamic learning rate adjustment of every process. It is important to note that this steps in overall were time consuming and required fine tuning of the code whenever runtime errors were encountered with a heavy requirement on the hardware resources such as the RAM, CPU and the GPU.

3.3. Faster R-CNN with Resnet50-FPN

We trained the Faster R-CNN model using the VisDrone dataset and the PyTorch implementation. Since the Faster R-CNN model wasn't initially trained to detect the precise things we were interested in or from the perspective of a drone, we had to fine-tune it to recognize these objects accurately. Despite this, we believed that with proper training, the Faster R-CNN model would be effective and efficient. This was due to it being a cutting-edge model that had already demonstrated its high performance in object detection. Ultimately, through the fine-tuning process, we were able to adapt the model to our specific use case, and it achieved impressive levels of accuracy in object detection. We trained Faster R-CNN over 10 epochs with the standard 480 image size. The model might struggle to accurately detect objects from the perspective of a drone, as it wasn't initially trained for this task. Also, The dataset might have class imbalances or other biases that could negatively impact the model's performance. During the training process of the Faster R-CNN model using the VisDrone dataset, we encountered some difficulties. The training process was computationally intensive, and the model required a high-end GPU to handle the load efficiently. Unfortunately, the GPU we had access to had limited memory, which led to some training instances failing due to memory errors. As a result, we had to optimize the training process and reduce the batch size to ensure that the training could continue without memory errors. Some of the issues with the image include a car in image that was detected as a van and identified as a container as a truck. Additionally, the model was unable to recognize the motors, which is a critical error. Further training of the Faster R-CNN model may help to identify these objects more accurately. Faster R-CNN with Resnet50 was effective in identifying the objects for which we trained it, but because of its imprecision, it falls short of the required standard for object identification in real-time.

3.4. Mask R-CNN

As part of my object detection project, We used the Mask RCNN PyTorch model to develop a model that could accurately detect various objects in aerial images from the VIsdrone 2019 dataset.

Mask R-CNN is an advanced method for identifying and separating objects that build on the Faster R-CNN design. It introduces a new pathway for forecasting object masks, which operates concurrently with the existing approach for identifying bounding boxes. This algorithm is extensively employed in the field of computer vision, particularly for detecting and segmenting objects in images and videos.

The VisDrone 2019 dataset is a benchmark dataset for object detection and tracking in unmanned aerial vehicle (UAV) applications. It contains high-resolution images and annotations of various objects, including people, vehicles,

and bicycles, in diverse scenarios and environments. The dataset offers a challenging and diverse testbed for evaluating object detection algorithms, and it has been used to advance state-of-the-art in computer vision research for UAV applications. The VisDrone 2019 dataset is widely used in academic and industrial communities, and it has helped to drive progress in the development of effective object detection and tracking techniques for UAVs.

We obtained the VIsdrone 2019 dataset and preprocessed it by removing any corrupted images and augmenting the remaining images to increase the size of the dataset.We also split the data into training and validation sets.

We trained the Mask RCNN model using PyTorch, with a learning rate of 0.005, a batch size of 4, and 5 epochs. I also used transfer learning with a pre-trained ResNet-50 backbone and fine-tuned the model using the VIsdrone dataset.

We evaluated the performance of the trained model using mean average precision (mAP) and intersection over union (IoU) metrics. The model achieved an mAP of 0.3 and an IoU of 0.5.

Overall, the results of the project demonstrate the effectiveness of the Mask RCNN PyTorch model for object detection in aerial images, and highlight the potential applications of this technology in fields such as remote sensing and surveillance. In future work, it may be possible to improve the performance of the model by using more advanced techniques such as multi-task learning or attention mechanisms.

One of the main challenges was a class imbalance, as the VIsdrone dataset had a significant imbalance between the number of object instances for different classes. To address this challenge, We used a combination of data augmentation and hyperparameter tuning techniques.

We anticipated that we would face issues on preprocessing Visdrone data as the dataset format was not in the usual PASCAL VOC or COCO. The annotations were in the Txt file, and we know that we would face issue.

AS anticipated we faced issue during pre-processing and training the data. We had to research and do multiple trials and error on the logic. To address additional challenges, we used a combination of data augmentation and hyperparameter tuning techniques. We also encountered issues during training our model as we were training more than 6000 images we had to change the image size and reduce the batch size as expected to our model

We tried was using a pre-trained Mask RCNN model from GitHub and it failed to work. We have to eventually use a pret-trained MASK RCNN model from pytorch in order to make it work.

3.5. YOLO v3

We utilized the YOLOv3 model from the Ultralytics implementation and trained it with the Visdrome dataset. Inherently, YOLOv3 was not trained with the purpose of identifying objects that we were interested in detecting or the perspective by which a drone captures objects, so we had to train the YOLOv3 so that it was capable of accurately recognizing them. We were confident that YOLOv3 would be accurate if we trained it, being once state-of-the-art, advanced models were bound to be both more effective and efficient. Because YOLOv3 was older than the newer models, it was bound to be lacking in features and updates, so I was anticipating it to run slower and be not as accurate due to data format inconsistencies. There were issues with training the model. As my hardware is not the most advance. Training large data sets for machine learning usually requires a graphic card with lots of VRAM, consequently, with only 12 GB of VRAM, sometimes during the training process, the model could not continue because of the VRAM limitation. Some poor accuracy that could be observed within an image was the van in the top left section of the image resulted in an accuracy of 31.8and a bicyclist that is completely missed near the center-left . Reasons for this include the car being white, and the zebra crosswalk also being white along with a light post overlapping it, misconstruing a clear vision of the vehicle. As for the bicyclist, while appearing to wear gray, is completely not recognized, which is a fatal error. It is possible that training YOLOv3 further can potentially make it identifiable. YOLOv3 was successful at detecting objects that we trained it for, but it does not meet an acceptable standard for real-time detection of objects with its imprecision.

3.6. YOLOv5

To implement yolov5, the open-source repository by ultralytics was used and can be found here: https://github.com/ultralytics/yolov5[3]. Now, the question remained how to leverage this architecture to our domain use-case to get the best performance possible. The straightforward answer here was to use transfer learning to get the best initialization weights for the model. To achieve this, we initialized the models with trained weights on the COCO dataset, which has 80 classes with whom most of our target classes overlap, like person, car, etc. This provides us with a great advantage in terms of knowledge transfer while also getting a really good initialization point for our model. Now to fine-tune the model to our domain use case or basically train the model to learn to detect the specific 12 classes we want, the only change in the architecture is to change the size of the softmax classification layer in the classification head to 12. Then the final problem is to tune the hyperparameters for the training to get the best performance possible in terms of mAP. This required rigorous experimentation by changing values for the learning rate, momentum for the optimizer, decay rate for the learning rate, etc. Due to computational resource constraints, search space was kept

small, but we were able to get competitive results. Early stopping training strategy was also used to prevent overfitting or diverged training. We trained two versions of yolov5, namely yolov5s and yolov5x, which stand for the small and extra-large versions of the model that differ in the number of hidden layers and consequently the number of parameters. The repository for the project contains training logs containing our finalized hyperparameter/config values inside `opt.yaml` file.

3.7. YOLOv8

We utilized the yolov8 implementation by ultralytics, an open-source repository, which can be accessed at the following URL: https://github.com/ultralytics/ultralytics/tree/main/ultralytics/yolo/v8. Our objective was to adapt this architecture to our specific domain use-case in order to achieve optimal performance.The most effective approach we discovered was to employ transfer learning to obtain the best initial weights for the model. We accomplished this by initializing the models with pre-trained weights from the COCO dataset, which consists of 80 classes, many of which overlap with our target classes such as person and car. This strategy provided us with a significant advantage in terms of knowledge transfer and served as an excellent starting point for our model.

To fine-tune the model for our specific domain use-case, which involved training it to detect our desired 12 classes, we made a simple change to the architecture by modifying the size of the softmax classification layer in the classification head to accommodate 12 classes.

I encountered several challenges during the model training process, primarily due to limitations in my hardware setup. Since training large datasets for machine learning typically requires a graphics card with ample VRAM, the 16 GB VRAM capacity of my hardware often posed a hindrance. At times, the training process was unable to continue due to these VRAM limitations. Furthermore, there were instances of poor accuracy in the model's predictions. For example, in some images, the accuracy for detecting a van located in the top left section was only 38.6. Additionally, there was a tricyclist near the center of the image that was completely missed by the model. Several factors contributed to these inaccuracies, including the car being white, the presence of white lines on the road, and a light post overlapping with the car, which hindered the model's ability to clearly perceive the vehicle.Regarding the tricyclist, despite wearing reflective clothing, the model failed to recognize them, which is a critical misinterpretation. It is possible that further training of YOLOv8 could potentially improve its ability to identify such cases.

The final challenge was to optimize the hyperparameters during training to achieve the best possible performance in terms of mAP (mean average precision). This necessitated extensive experimentation, involving adjustments to parameters such as the learning rate, momentum for the optimizer, decay rate for the learning rate, and so on. Due to limitations in computational resources, we had to limit the search space for hyperparameter tuning. Nevertheless, we were able to obtain competitive results. We also employed the early stopping training strategy to prevent overfitting or diverged training.We trained two versions of the yolov8 model: yolov8s (small) and yolov8x (extra-large). These versions differ in the number of hidden layers and consequently the number of parameters.

4. Evaluation

When evaluating the accuracy and consistency of an object detection model, we can use quantitative, qualitative, and composite metrics.

4.1. Quantitative Metrics

Quantitative metrics include mAP, precision, recall, PR curve, and AUC curve, which are used to measure the accuracy and consistency of the model in detecting objects across multiple categories. The mAP score is calculated by measuring the area under the precision-recall curve, while precision is the fraction of correctly predicted positive samples out of all positive predictions made by the model. Recall is the fraction of correctly predicted positive samples out of all actual positive samples present in the dataset. PR curve is a graphical representation of the precision-recall trade-off for different thresholds used in the model, while AUC curve is used to measure the performance of the model based on its ability to distinguish between positive and negative samples.

4.2. Qualitative Metrics

Qualitative metrics include visual plotting of predictions on input images, which helps in visualizing the performance of the model across different objects.

4.3. Composite Metrics

Composite metrics include F1 score/curve and confusion matrix. F1 score is the harmonic mean of precision and recall and provides a balanced measure of model performance across precision and recall. Confusion matrix is a table that summarizes the number of true positives, false positives, true negatives, and false negatives predicted by the model and provides a detailed view of the model's performance across different categories.

These evaluation metrics are useful in assessing the accuracy and consistency of an object detection model and providing insights into its performance across different categories and thresholds.

5. Results

The PR curve helps us see how well a model can find the right objects and avoid mistakes. A high-precision model makes fewer mistakes while a high-recall model finds most of the objects. We use the PR curve to compare the performance of the models at different levels of accuracy, and we can measure this performance using the AUC score. We will now look at the PR curves for each of the models.

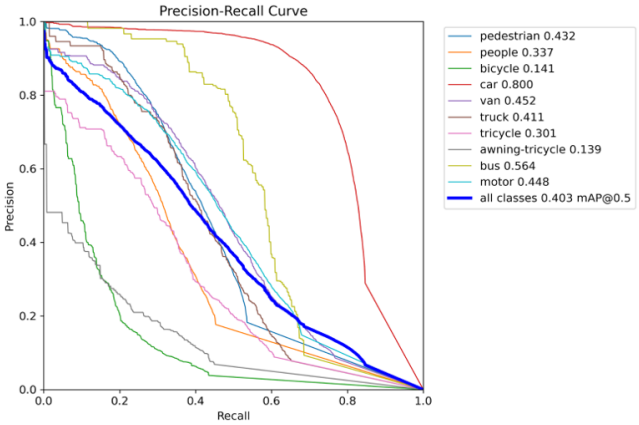


Figure 1. Precision-Recall Curve for YOLO v3

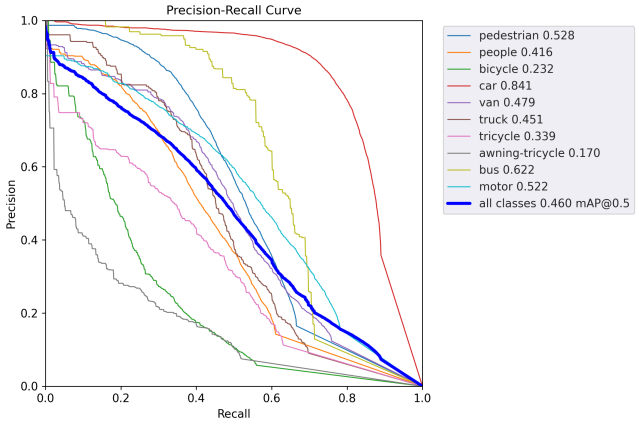


Figure 2. Precision-Recall Curve for YOLO v5s (small)

The table1 shows mAP@0.5 and Inference Time(FPS), The mean Average Precision (mAP) is a commonly used metric to evaluate object detection models. This table provides additional information on the performance of each model, specifically their mAP@0.5 and inference time in frames per second (FPS), which can be helpful in deciding which model to use in different scenarios.

6. Conclusion

In conclusion, we compared several object detection models using the mean average precision (mAP) metric at a threshold of 0.5 and inference time (FPS). The YOLO-v8 Extra-Large model achieved the highest mAP score of 0.56, while the YOLO-v8 Small model had the fastest inference time of 1.20 milliseconds. However, it's important to consider the trade-off between accuracy and speed when

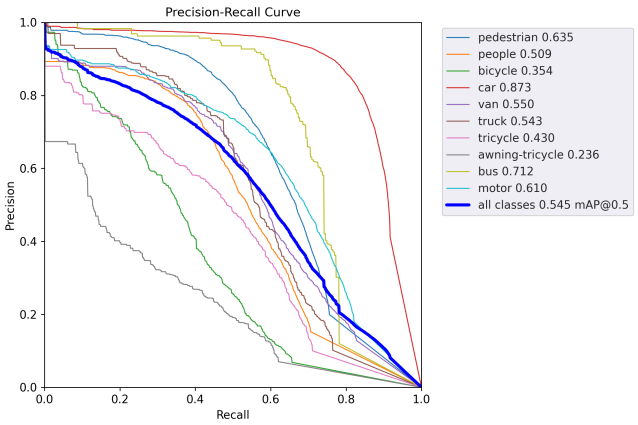


Figure 3. Precision-Recall Curve for YOLO v5x (extra large)

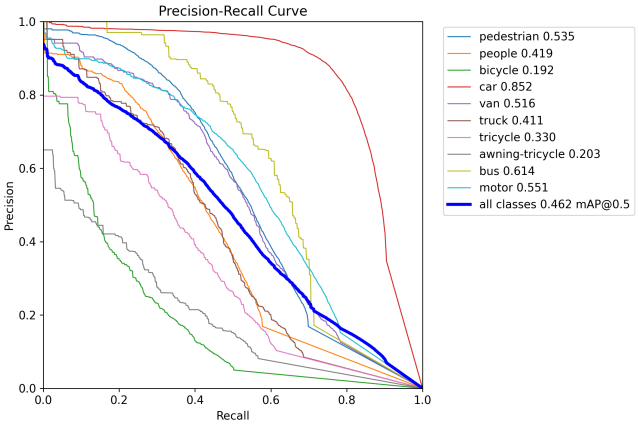


Figure 4. Precision-Recall Curve for YOLO V8s (small)

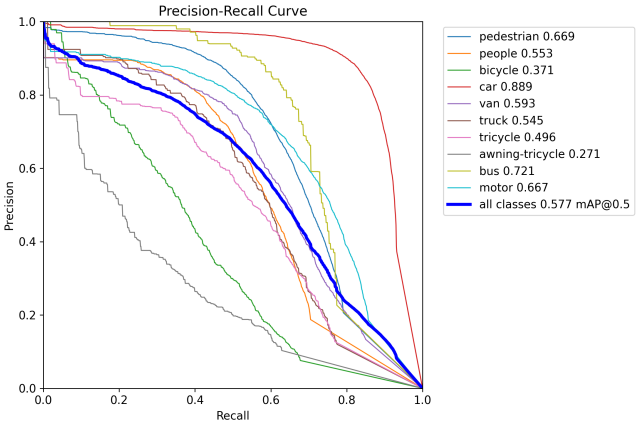


Figure 5. Precision-Recall Curve for YOLO v8x (extra large)

Algorithm	mAP@0.5	Inference Time (ms)
SSD	0.12	30
RCNN with Resnet-50	0.30	190
Faster-RCNN (Resnet50)	0.23	150
Faster-RCNN (Resnet50-FPN	0.27	170
Mask-RCNN	0.29	200
Yolo-v3	0.40	22
Yolo-v5-Small	0.46	6.4
Yolo-v5-Extra-Large	0.54	12.1
Yolo-v8-Small	0.46	**1.20**
Yolo-v8-Extra-Large	**0.56**	3.53

Table 1. Table showing mAP@0.5 and FPS of each models

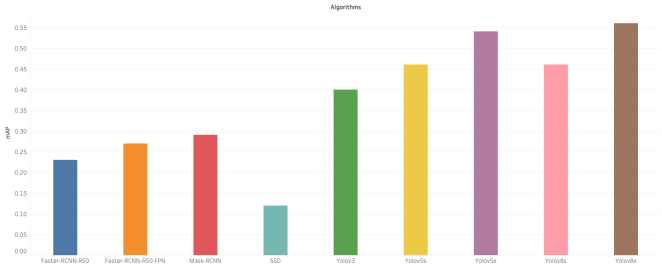


Figure 6. mAP Comparison between Algorithms.

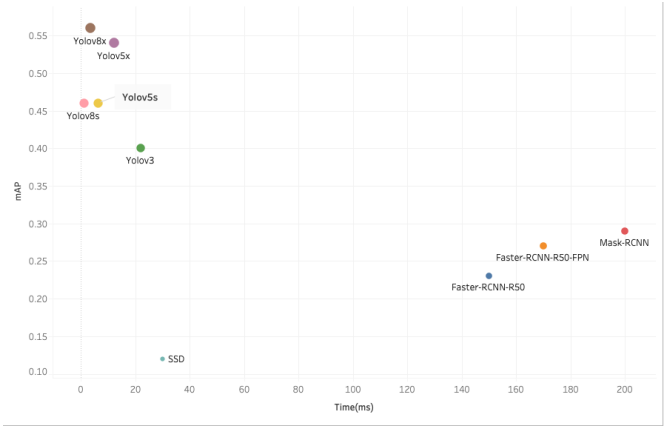


Figure 7. mAP V/s Inference Time Comparison for all the Models.

selecting a model for a specific use case.

7. Work Division

We acknowledge the contributions of the following individuals to the project. Each member brought unique skills and expertise to the team, resulting in the successful completion of the project:

Nimesh Arora (Team Lead): Led the comparitive study: Created Dataloader for Vizdrone; Implementatied and trained YoloV8 small and extra Large versions on an HPC, and created notebooks to fetch logs to plot results and metrics for all models. Covered log-loss curves, assisted fellow teammates in resolving issues and solving bugs in model implementation, and prepared a demo for the presentation.

Ashish Gurung: Data Creation and Implementation. Analyzed the outcomes of training and deploying Faster RCNN on the Visdrone dataset using PyTorch, a real-time object identification method.

Kavya Reddy: Trained and implemented SSD using pytortch using Visdrone Dataset.

Dikshant Sagar: Data Creation and Implementation. Trained the Yolov5 small version on Vizdrone dataset with 50 epochs and graphed the results.

Ismael Valenzuela: Trained and implemented YOLOv3 with the Visdrone dataset, the real-time object detection algorithm, and analyzed the results.

Safal Rijal: Contributed to research by collecting information from academic articles, writing reports, and creating presentation materials.

Shreyas Teli: Data Creation and Implementation. Implemented and trained Mask-RCNN, calculated train loss, accuracy, and mAP.

Harika Yerramalli: Trained the faster-rcnnresnet-50FPN model using VisDroneDet2019-train dataset validated it using VisDroneDet2019val and added non-maximum suppression(nms)to remove duplicate detections. A selective search algorithm was also implemented so as to generate region proposals.

References

[1] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014. 2

[2] Glenn Jocher, Ayush Chaurasia, Alex Stoken, Jirka Borovec, Yonghye Kwon, Kalen Michael, Jiacong Fang, Zeng Yifu, Colin Wong, Diego Montes, et al. ultralytics/yolov5: v7. 0-yolov5 sota realtime instance segmentation. *Zenodo*, 2022. 3

[3] Glenn Jocher, Chien-Yao Wang, and Alexander Amini. Yolov5: Improved real-time object detection. In *NeurIPS Workshop on Efficient Deep Learning for Computer Vision*, 2020. 3, 5

[4] Martin Meyer, Arber Zela, Saad Naveed, Issam H Laradji, Mohammad J Shafiee, Judy Hoffman, and Heng Zhang. Yolov8: You only look once for efficient object detection. *arXiv preprint arXiv:2208.00519*, 2022. 3

[5] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 779–788, 2016. 3

[6] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5574–5583. IEEE, 2018. 3

[7] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Neural Information Processing Systems*, pages 91–99, 2015. 2

[8] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision*, pages 818–833, 2014. 2

[9] Chen Zhao, Wenqian Xu, Yong Liu, and Junjie Yan. Self-supervised feature learning for object detection in surveillance videos. *IEEE Transactions on Industrial Informatics*, 17(8):5895–5905, 2021. 2

[10] C Lawrence Zitnick and Piotr Dollár. Edge boxes: Locating object proposals from edges. In *European Conference on Computer Vision*, pages 391–405, 2014. 1