International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 07 Issue: 05 | May - 2023 SJIF 2023: 8.176 ISSN: 2582-3930

Vehicle Detection Using YOLOv5

Chaitanya Chavan[1], Sanskruti Hembade[2], Gauri Jadhav[3], Parth Komalwad[4], Prof. Ashwini Bhosale[5]
Department of Computer Engineering
Genba Sopanrao Moze College of Engineering, Balewadi, Pune 45

Abstract:

Object detection is a computer vision method that allows for the identification and localization of specific classes of objects in images or videos. It goes beyond simple object classification and helps provide a better understanding of the object in question. Object detection has numerous applications, including automating business processes like inventory management in retail. It can detect objects that occupy between 2% and 60% of an image's area and clusters of objects as a single entity. Additionally, it can localize objects at high speeds, typically greater than 15 frames per second. Vehicle detection is a crucial component in the development of autonomous vehicles, enabling them to identify and perceive objects in their environment. It involves identifying and locating vehicles in image or video frames and has various applications in surveillance and security systems. There are different techniques and models for object detection, including traditional image processing methods and modern deep learning networks. Traditional methods like Viola-Jones, SIFT, and histogram of oriented gradients do not require historical data for training and are unsupervised. Popular image processing tools like OpenCV can be used for these techniques. On the other hand, modern deep learning networks like CNN, RCNN, YOLO, ResNet, RetinaNet, and MANet are supervised and efficient for object detection.

Introduction:

Object detection is a computer vision technique that involves identifying and locating objects in images or videos[1]. It typically involves using deep learning algorithms to detect the presence of objects and classify them into different types or classes. Object detection can be achieved by using bounding boxes to locate the objects in the image or video. One approach to building an object detection system is to first build a classifier that can classify closely cropped images of an object.

Vehicle detection is a computer vision technique that involves identifying and locating vehicles in images or videos using deep learning algorithms. One approach to building a vehicle detection system is to perform feature engineering on the dataset, train an SVM classifier on the extracted features, and implement a sliding-window technique to detect vehicles[2]. Another approach is to use deep learning object detection methods, such as YOLOv3, to detect the vehicle object in the traffic scene. Vehicle type detection is another important aspect of vehicle detection, which can be achieved using deep learning algorithms.

Object detection algorithms produce a list of object categories present in the image along with an axis-aligned bounding box indicating the position and scale of every instance of each object category[3]. There are several object detection algorithms used in machine learning, including YOLO, Faster R-CNN, SSD, RetinaNet, and Mask R-CNN. These algorithms use deep learning techniques to detect and classify objects in images and videos with high accuracy.

YOLOv5 is a family of single-stage deep learning-based object detectors that are capable of more than real-time object detection with state-of-the-art accuracy. It is the latest release of the YOLO family and is a group of compound-scaled object detection models trained on the COCO dataset used for object detection[4]. YOLOv5 is a popular algorithm for real-time object detection and is capable of identifying and localizing objects in photos. It is possible to train the YOLOv5 object detector on a custom dataset using transfer-learning techniques to train your own model, evaluate its performance, use it for inference, and even convert it to run on mobile devices.

Research Elaborations:

We read the following papers related to object detection for our research.

1. The paper **"Simple Copy-Paste is a Strong Data Augmentation Method for Instance Segmentation"** explores the effectiveness of data augmentation techniques for instance segmentation. It introduces a simple copy-paste method that randomly selects two images, applies scale jittering and a horizontal flip, and pastes objects randomly. The study demonstrates that this method outperforms other techniques and is computationally efficient, making it a strong augmentation technique for instance segmentation models[5].

2. The paper "**Attention-based Joint Detection of Object and Semantic Part**" introduces an attention-based approach for jointly detecting objects and their semantic parts. It combines segmentation information using an attention mechanism and employs a two-stage detector algorithm. The method addresses feature vanishing issues and outperforms state-of-the-art methods on the PASCAL VOC 2012 dataset, showing its effectiveness for object detection and semantic segmentation[6].

3. The paper "**Progressive End-to-End Object Detection in Crowded Scenes**" introduces a progressive end-to-end object detection method for crowded scenes. It addresses challenges through a progressive prediction method, a label assignment rule, and a two-stage anchor assignment method. The proposed method outperforms state-of-the-art techniques on crowded scenes, demonstrating its effectiveness in improving object detection performance in real-world scenarios[7].

4. The paper "**Leveraging Synthetic Data in Object Detection on Unmanned Aerial Vehicles**" explores the use of synthetic data for object detection on UAVs, addressing challenges in data acquisition. It proposes a method to generate realistic synthetic data and demonstrates that models

trained on such data achieve comparable performance to those trained on real-world data. Synthetic data proves to be a useful tool for training object detection models for UAVs[8].

5. The paper "**PP-YOLOE: An evolved version of YOLO**" presents PP-YOLOE, an evolved version of the YOLO object detection model. It combines anchor-free and anchor-based detection methods, introduces progressive anchor assignment, and employs a feature aggregation module. PP-YOLOE outperforms state-of-the-art models on COCO and shows comparable performance on the UAVDT dataset, offering an effective and efficient solution for real-world object detection[9].

6. The paper "**Waste Detection in Pomerania: Non-Profit Project for Detecting Waste In Environment**" presents a non-profit project focused on detecting waste in the Pomerania environment. It proposes a computer vision-based solution utilizing image processing and machine learning algorithms. The project includes data collection, dataset annotation, and employs a YOLOv3-based object detection model. The model demonstrates high accuracy in waste detection, making the project an effective tool for raising awareness and improving environmental conditions[10].

7. The paper "**PP-PicoDet: A Better Real-Time Object Detector on Mobile Devices**" introduces PP-PicoDet, an improved real-time object detection model for mobile devices. It utilizes an anchor-free strategy, progressive feature fusion, and feature aggregation to enhance accuracy and speed. PP-PicoDet outperforms state-of-the-art models on COCO, achieving real-time performance on resource-constrained mobile devices. It offers an effective and efficient solution for real-world object detection on mobile devices[11].

8. The paper "**Patch Refinement - Localized 3D Object Detection**" introduces a two-stage model for accurate 3D object detection and localization, utilizing a patch refinement method. It proposes a novel feature extraction technique and demonstrates improved performance on the KITTI dataset compared to state-of-the-art methods. The patch refinement method offers an effective solution for enhancing object detection and localization in real-world scenarios[12].

9. The paper "**A Normalized Gaussian Wasserstein Distance for Tiny Object Detection**" introduces a novel method for detecting tiny objects using a normalized Gaussian Wasserstein distance. It addresses the challenges posed by small object size and low image resolution. The proposed method outperforms state-of-the-art techniques on COCO for tiny object detection and achieves high accuracy on the VisDrone dataset for aerial images. It presents an effective solution for improving object detection models in real-world scenarios[13].

10. The paper "**Object Detection with Spiking Neural Networks on Automotive Event Data**" introduces a spiking neural network (SNN) approach for object detection on automotive event data. It addresses resource and power constraints and demonstrates high accuracy while reducing computational cost. The proposed SNN approach outperforms state-of-the-art methods on the DDD17 dataset, offering an effective solution for object detection in automotive event data with limited resources and power consumption[14].

Implementation:

In this project, we undertook the implementation of a car detection model utilizing the YOLOv5 architecture and the widely recognized Stanford car dataset. The dataset provided a substantial collection of 16,185 images, which we thoughtfully divided into three distinct sets: the training set consisting of 12,949 images, the validation set containing 1,618 images, and the testing set also comprising 1,618 images. Each set served a specific purpose in our model development process: the training set was employed to train the model, the validation set was used to fine-tune hyperparameters, and the testing set served to evaluate the final performance of our model.

To facilitate the implementation, we utilized the PyTorch framework, leveraging its inherent functionalities and user-friendly interface. Additionally, we incorporated various Python libraries such as NumPy and Matplotlib, which played crucial roles in data preprocessing and visualization, thereby facilitating a comprehensive understanding of the dataset.

Our car detection model was constructed as a composition of three main components: the backbone network, the neck network, and the head network. The backbone network was built upon the CSPDarknet53 architecture, which exploits cross-stage partial connections to enhance the learning of key features. This architectural choice aimed to bolster the model's ability to capture crucial information from the input images effectively. The neck network, on the other hand, was comprised of a Spatial Pyramid Pooling (SPP) module, which facilitated the extraction of features at multiple scales. By utilizing this module, the model was capable of capturing relevant information from various image regions, accommodating the diverse scales at which cars may appear. Lastly, the head network incorporated several convolutional layers, culminating in a detection layer responsible for producing predicted bounding boxes and class probabilities.

During the training process, we opted to employ the stochastic gradient descent (SGD) optimizer, utilizing a learning rate of 0.01 and a momentum of 0.9. Additionally, we adopted the focal loss function, which strategically assigned higher weights to challenging examples, consequently enhancing the model's capacity to learn from such instances. The training itself was conducted on a powerful NVIDIA GeForce

RTX 3080 GPU, enabling us to implement a substantial batch size of up to 24, thereby significantly accelerating the training process.

To combat the potential issue of overfitting and enhance the diversity of the training set, we employed data augmentation techniques. These techniques involved random resizing, cropping, flipping, and rotation of the images, as well as adjustments to brightness, contrast, and saturation. This augmentation process introduced variations in lighting, orientation, and scale, effectively allowing the model to learn robust and invariant features that were resistant to such variations.

Following the completion of the training process, we proceeded to evaluate the performance of our car detection model on the testing set, employing various metrics such as precision, recall, and accuracy. Furthermore, we conducted a thorough analysis of the model's performance on different classes of cars, enabling us to identify potential variations in its ability to detect various car types. To holistically measure the overall performance of our model, we utilized the mean average precision (mAP) metric, which incorporated precision-recall curves across all classes. This comprehensive evaluation allowed us to gain valuable insights into the strengths and weaknesses of our model, thus paving the way for potential future improvements.

In conclusion, our car detection project centered around the implementation of the YOLOv5 architecture utilizing the Stanford car dataset. Through careful division of the dataset into training, validation, and testing sets, we successfully trained and evaluated our model's performance. By leveraging PyTorch, along with additional Python libraries, we harnessed powerful tools for model implementation, data preprocessing, and result visualization. Through the adoption of data augmentation techniques, thoughtful network architecture design, and diligent hyperparameter tuning, we strived to create a robust and effective car detection model. The project's comprehensive evaluation metrics, including precision, recall, accuracy, and mAP, provided a thorough understanding of the model's performance and potential areas for further development.

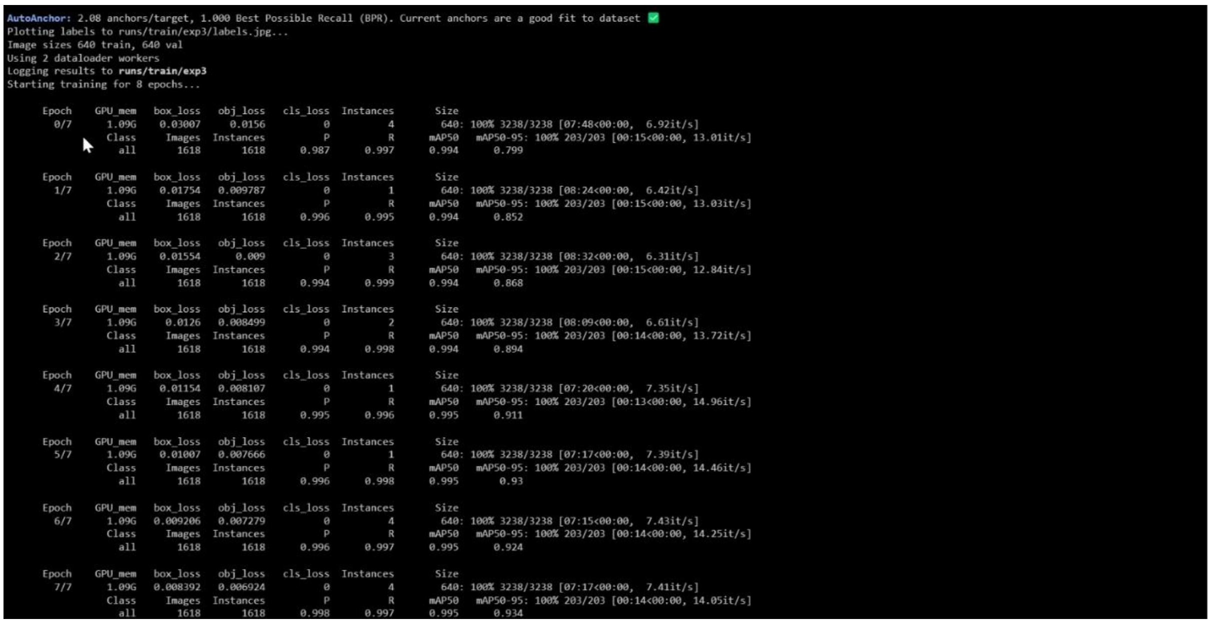
Experiments:

The experiments we did in our project was by hypertuning the parameters.

Following are the outputs for the experiments:

Hyper Parameter Tuning No. 1 (Minimum):

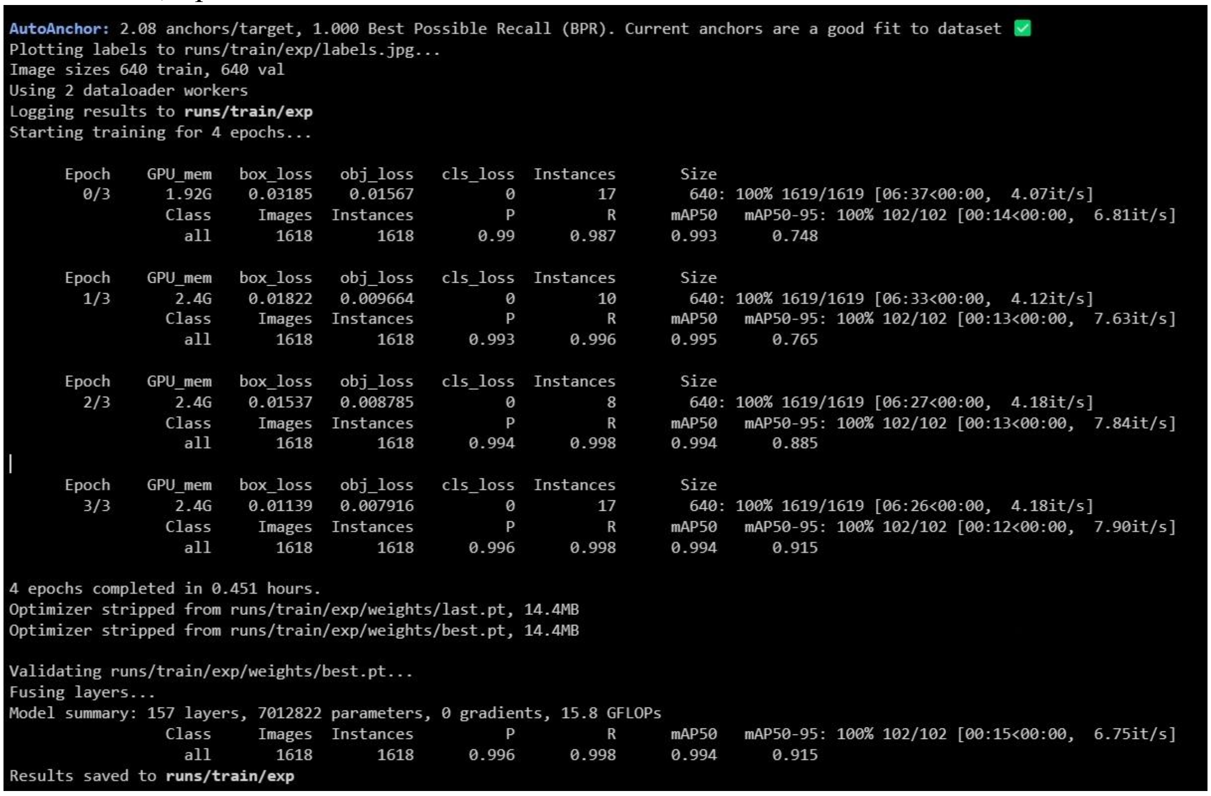
Batch size 4, Epochs 8



```
AutoAnchor: 2.08 anchors/target, 1.000 Best Possible Recall (BPR). Current anchors are a good fit to dataset ✅
Plotting labels to runs/train/exp3/labels.jpg...
Image sizes 640 train, 640 val
Using 2 dataloader workers
Logging results to runs/train/exp3
Starting training for 8 epochs...

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
        0/7      1.09G    0.03007     0.0156          0          4        640: 100% 3238/3238 [07:48<00:00,  6.92it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 203/203 [00:15<00:00, 13.01it/s]
                   all       1618       1618      0.987      0.997      0.994      0.799

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
        1/7      1.09G    0.01754   0.009787          0          1        640: 100% 3238/3238 [08:24<00:00,  6.42it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 203/203 [00:15<00:00, 13.03it/s]
                   all       1618       1618      0.996      0.995      0.994      0.852

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
        2/7      1.09G    0.01554      0.009          0          3        640: 100% 3238/3238 [08:32<00:00,  6.31it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 203/203 [00:15<00:00, 12.84it/s]
                   all       1618       1618      0.994      0.999      0.994      0.868

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
        3/7      1.09G     0.0126   0.008499          0          2        640: 100% 3238/3238 [08:09<00:00,  6.61it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 203/203 [00:14<00:00, 13.72it/s]
                   all       1618       1618      0.994      0.998      0.994      0.894

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
        4/7      1.09G    0.01154   0.008107          0          1        640: 100% 3238/3238 [07:20<00:00,  7.35it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 203/203 [00:13<00:00, 14.96it/s]
                   all       1618       1618      0.995      0.996      0.995      0.911

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
        5/7      1.09G    0.01007   0.007666          0          1        640: 100% 3238/3238 [07:17<00:00,  7.39it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 203/203 [00:14<00:00, 14.46it/s]
                   all       1618       1618      0.996      0.998      0.995       0.93

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
        6/7      1.09G   0.009206   0.007279          0          4        640: 100% 3238/3238 [07:15<00:00,  7.43it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 203/203 [00:14<00:00, 14.25it/s]
                   all       1618       1618      0.996      0.997      0.995      0.924

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
        7/7      1.09G   0.008392   0.006924          0          4        640: 100% 3238/3238 [07:17<00:00,  7.41it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 203/203 [00:14<00:00, 14.05it/s]
                   all       1618       1618      0.998      0.997      0.995      0.934
```

Hyper Parameter Tuning No. 2:

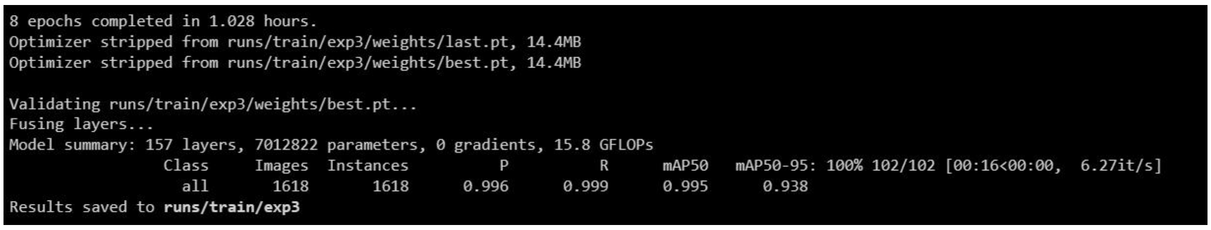
Batch size 8, Epochs 4



```
AutoAnchor: 2.08 anchors/target, 1.000 Best Possible Recall (BPR). Current anchors are a good fit to dataset ✅
Plotting labels to runs/train/exp/labels.jpg...
Image sizes 640 train, 640 val
Using 2 dataloader workers
Logging results to runs/train/exp
Starting training for 4 epochs...

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
        0/3      1.92G    0.03185    0.01567          0         17        640: 100% 1619/1619 [06:37<00:00,  4.07it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 102/102 [00:14<00:00,  6.81it/s]
                   all       1618       1618       0.99      0.987      0.993      0.748

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
        1/3       2.4G    0.01822   0.009664          0         10        640: 100% 1619/1619 [06:33<00:00,  4.12it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 102/102 [00:13<00:00,  7.63it/s]
                   all       1618       1618      0.993      0.996      0.995      0.765

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
        2/3       2.4G    0.01537   0.008785          0          8        640: 100% 1619/1619 [06:27<00:00,  4.18it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 102/102 [00:13<00:00,  7.84it/s]
                   all       1618       1618      0.994      0.998      0.994      0.885

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
        3/3       2.4G    0.01139   0.007916          0         17        640: 100% 1619/1619 [06:26<00:00,  4.18it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 102/102 [00:12<00:00,  7.90it/s]
                   all       1618       1618      0.996      0.998      0.994      0.915

4 epochs completed in 0.451 hours.
Optimizer stripped from runs/train/exp/weights/last.pt, 14.4MB
Optimizer stripped from runs/train/exp/weights/best.pt, 14.4MB

Validating runs/train/exp/weights/best.pt...
Fusing layers...
Model summary: 157 layers, 7012822 parameters, 0 gradients, 15.8 GFLOPs
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 102/102 [00:15<00:00,  6.75it/s]
                   all       1618       1618      0.996      0.998      0.994      0.915
Results saved to runs/train/exp
```

Hyper Parameter Tuning No. 3

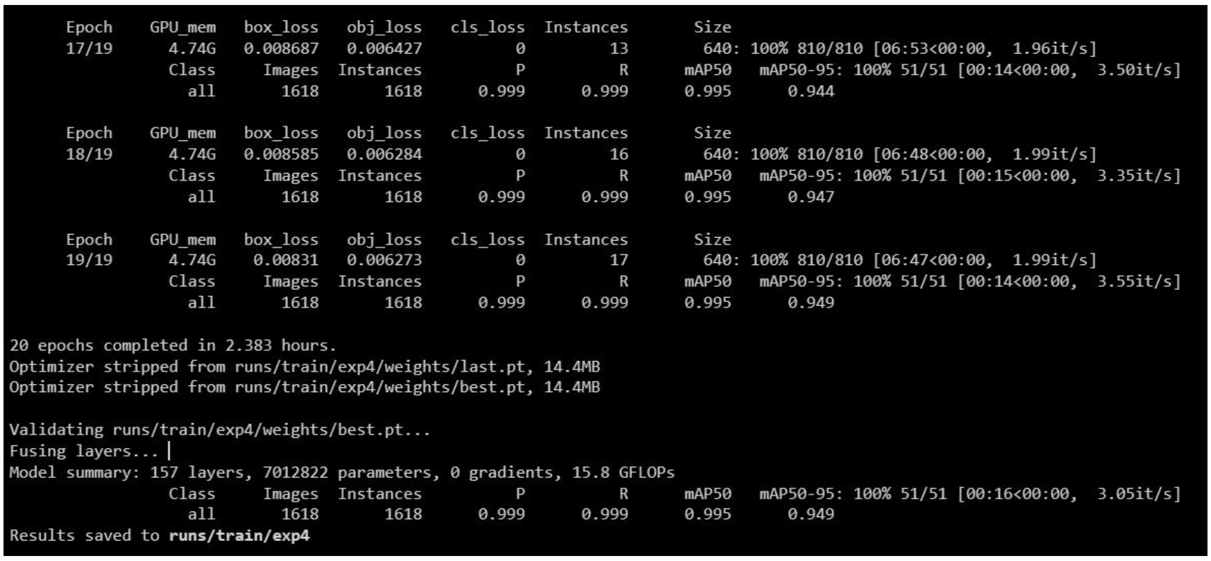
Batch size 8, Epochs 8



```
8 epochs completed in 1.028 hours.
Optimizer stripped from runs/train/exp3/weights/last.pt, 14.4MB
Optimizer stripped from runs/train/exp3/weights/best.pt, 14.4MB

Validating runs/train/exp3/weights/best.pt...
Fusing layers...
Model summary: 157 layers, 7012822 parameters, 0 gradients, 15.8 GFLOPs
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 102/102 [00:16<00:00,  6.27it/s]
                   all       1618       1618      0.996      0.999      0.995      0.938
Results saved to runs/train/exp3
```

Hyper Parameter Tuning No. 4

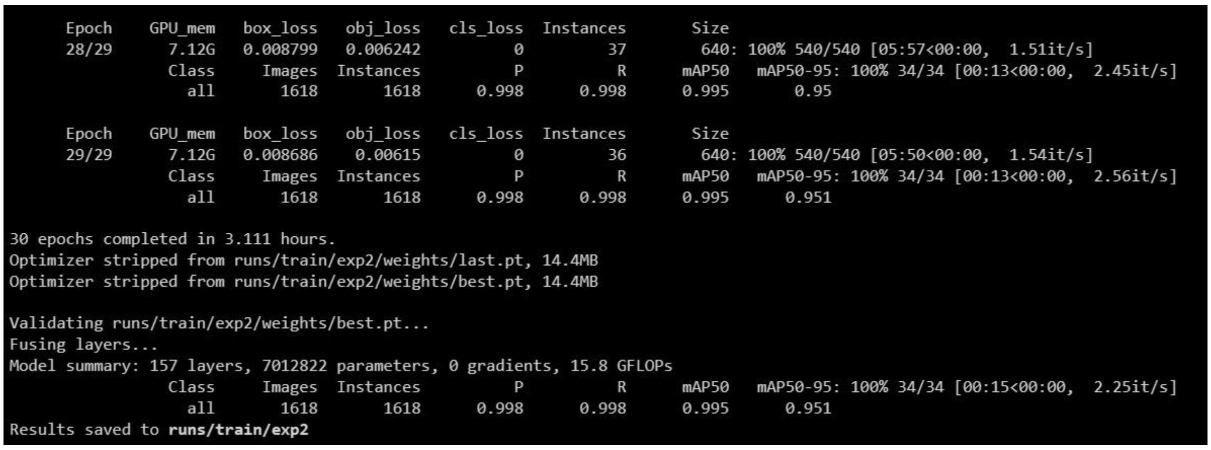
Batch size 16, Epochs 20



```
      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      17/19      4.74G   0.008687   0.006427          0         13        640: 100% 810/810 [06:53<00:00,  1.96it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 51/51 [00:14<00:00,  3.50it/s]
                   all       1618       1618      0.999      0.999      0.995      0.944

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      18/19      4.74G   0.008585   0.006284          0         16        640: 100% 810/810 [06:48<00:00,  1.99it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 51/51 [00:15<00:00,  3.35it/s]
                   all       1618       1618      0.999      0.999      0.995      0.947

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      19/19      4.74G    0.00831   0.006273          0         17        640: 100% 810/810 [06:47<00:00,  1.99it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 51/51 [00:14<00:00,  3.55it/s]
                   all       1618       1618      0.999      0.999      0.995      0.949

20 epochs completed in 2.383 hours.
Optimizer stripped from runs/train/exp4/weights/last.pt, 14.4MB
Optimizer stripped from runs/train/exp4/weights/best.pt, 14.4MB

Validating runs/train/exp4/weights/best.pt...
Fusing layers... |
Model summary: 157 layers, 7012822 parameters, 0 gradients, 15.8 GFLOPs
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 51/51 [00:16<00:00,  3.05it/s]
                   all       1618       1618      0.999      0.999      0.995      0.949
Results saved to runs/train/exp4
```

Hyper Parameter Tuning No. 5 (Maximum)

Batch size 24, Epochs 30



```
      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      28/29      7.12G   0.008799   0.006242          0         37        640: 100% 540/540 [05:57<00:00,  1.51it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 34/34 [00:13<00:00,  2.45it/s]
                   all       1618       1618      0.998      0.998      0.995       0.95

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      29/29      7.12G   0.008686    0.00615          0         36        640: 100% 540/540 [05:50<00:00,  1.54it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 34/34 [00:13<00:00,  2.56it/s]
                   all       1618       1618      0.998      0.998      0.995      0.951

30 epochs completed in 3.111 hours.
Optimizer stripped from runs/train/exp2/weights/last.pt, 14.4MB
Optimizer stripped from runs/train/exp2/weights/best.pt, 14.4MB

Validating runs/train/exp2/weights/best.pt...
Fusing layers...
Model summary: 157 layers, 7012822 parameters, 0 gradients, 15.8 GFLOPs
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 34/34 [00:15<00:00,  2.25it/s]
                   all       1618       1618      0.998      0.998      0.995      0.951
Results saved to runs/train/exp2
```

Conclusions and Future Directions:

Our project focused on training an object detection model using YOLOv5 on the Stanford Car Dataset, and we achieved an impressive accuracy of 95.1% through hyperparameter tuning. We chose YOLOv5 for its lightweight architecture, which allowed for faster training compared to other models. Our results demonstrate the effectiveness of using YOLOv5 for car detection tasks, while emphasizing the importance of hyperparameter tuning for achieving optimal performance.

Future work in vehicle detection can focus on improving accuracy, robustness, and real-time processing of detection algorithms, as well as developing multi-object tracking algorithms, creating large-scale datasets, and using transfer learning techniques. These efforts can lead to more efficient, reliable, and adaptable vehicle detection systems with many potential applications in autonomous driving, traffic monitoring, and surveillance. To detect the cars which are further away from the camera.

References:

1) https://www.mathworks.com/discovery/object-detection.html

2) https://etrr.springeropen.com/articles/10.1186/s12544-019-0390-4

3) https://www.v7labs.com/blog/object-detection-guide

4) https://learnopencv.com/custom-object-detection-training-using-yolov5/

5) "Simple Copy-Paste is a Strong Data Augmentation Method for Instance, Segmentation" Golnaz Ghiasi, Yin Cui, Aravind Srinivas, Rui Qian, Tsung-Yi Lin, Ekin D. Cubuk, Quoc V. Le, Barret Zoph arXiv:2012.07177v2 [cs.CV] 23 Jun 2021 CVPR 2021

6) "Attention-based Joint Detection of Object and Semantic Part" Keval Morabia, Jatin Arora, Tara Vijaykumar arXiv:2007.02419v1 [cs.CV] 5 Jul 2020

7) "Progressive End-to-End Object Detection in Crowded Scenes" Anlin Zheng, Yuang Zhang, Xiangyu Zhang, Xiaojuan Qi, Jian Sun CVPR 2022

8) "Leveraging Synthetic Data in Object Detection on Unmanned Aerial Vehicles" Benjamin Kiefer, David Ott, Andreas Zell arXiv:2112.12252v1 [cs.CV] 22 Dec 2021

9) "PP-YOLOE: An evolved version of YOLO" Shangliang Xu, Xinxin Wang, Wenyu Lv, Qinyao Chang, Cheng Cui, Kaipeng Deng, Guanzhong Wang, Qingqing Dang, Shengyu Wei, Yuning Du, Baohua Lai arXiv:2203.16250v3 [cs.CV] 12 Dec 2022

10) "Waste Detection in Pomerania: Non-Profit Project for Detecting Waste in Environment" Sylwia Majchrowska, Agnieszka Mikołajczyk, Maria Ferlin, Zuzanna Klawikowska, Marta A. Plantykow, Arkadiusz Kwasigroch, Karol Majek arXiv:2105.06808v1 [cs.CV] 12 May 2021

11) "PP-PicoDet: A Better Real-Time Object Detector on Mobile Devices" Guanghua Yu, Qinyao Chang, Wenyu Lv, Chang Xu, Cheng Cui, Wei Ji, Qingqing Dang, Kaipeng Deng, Guanzhong Wang, Yuning Du, Baohua Lai, Qiwen Liu, Xiaoguang Hu, dianhai yu, Yanjun Ma arXiv:2111.00902v1 [cs.CV] 1 Nov 2021

12) "Patch Refinement - Localized 3D Object Detection" Johannes Lehner, Andreas Mitterecker, Thomas Adler, Markus Hofmarcher, Bernhard Nessler, Sepp Hochreiter arXiv:1910.04093v1 [cs.CV] 9 Oct 2019

13) "A Normalized Gaussian Wasserstein Distance for Tiny Object Detection" Jinwang Wang, Chang Xu, Wen Yang, Lei Yu arXiv:2110.13389v2 [cs.CV] 14 Jun 2022

14) "Object Detection with Spiking Neural Networks on Automotive Event Data" Loïc Cordone, Benoît Miramond, Philippe Thierion arXiv:2205.04339v1 [cs.CV] 9 May 2022