UNDERSTANDING OBJECT DETECTION'S ROLE IN DECISION-MAKING FOR SELF-DRIVING VEHICLES

Shishir Bahubali

ABSTRACT

Our research aimed to understand computer vision in self-driving vehicles through object detection. We evaluated different AI models to determine the best performer for this task. Self-driving cars, led by companies like Tesla, rely heavily on advanced software, and understanding these systems is crucial before their widespread adoption. We applied machine learning models to detect vehicles in highway surveillance footage, comparing their precision and recall. Among the models tested, VGG16 outperformed others, achieving a precision of 0.7209 and a recall of 0.7298. Object detection in self-driving cars identifies vehicles and other road objects, ensuring safe navigation. Achieving high precision and recall requires large datasets with tens of thousands of images and thorough hyperparameter tuning. Choosing the right model is critical; in our case, VGG16 proved the most effective. These findings underscore the importance of robust model selection and optimization in advanced self-driving technology.

INTRODUCTION

Our research question is: how can object detection improve the decision-making of self-driving cars? This question is important because companies such as Tesla are pioneering self-driving technology every day, however, it is important to understand the limitations of this technology and educate the public on its workings. Object detection utilizes AI models to identify objects in images. This is crucial for self-driving cars as they need to spot and avoid objects to ensure passenger safety. To understand object detection and its role in improving decision-making in self-driving cars we implemented 3 different object detection algorithms and compared 3 different types of machine learning models. Each model used supervised learning, labeling images of vehicles with their respective types. Since this problem involved classification, we used VGG16, YOLOV8, and CNN models in our research. The data used comprised visual footage of vehicles from highway surveillance [1]. Each image was labeled with the type of vehicle, such as trucks, cars, motorbikes, etc. After preprocessing the data, we trained and tested each model. This was done by splitting the data into training and testing sets. The output metrics, such as accuracy, were used to compare each model.

BACKGROUND

Three articles we used before conducting the coding part of our research were, “Deep learning for object detection and scene perception in self-driving cars: Survey, challenges, and open issues”, “What self-driving cars tell us about AI risks”, and “The road to fully self-driving car transport may be perilous”. In the first article, we learned more about how self-driving cars use object detection and some of the technology's pros and cons. We learned that the technology has limitations such as even with the most optimal algorithm the cars still need human assistance as they struggle in harsh terrain conditions and that the cars can’t identify different environments on their own. By understanding these limitations, we further comprehend the significant improvements that need to be made to object detection technology to improve the effectiveness of self-driving cars [1]. In the second article, we began to understand the improvements needed as consumers regarding self-driving cars. Instead of just being in awe of this revolution, we also need to understand its limitations. The technology isn’t at a point where we can trust it with our lives, passengers need to be vigilant to ensure the car doesn’t malfunction. By improving our understanding, we can better facilitate the improvement of the technology by reducing the number of accidents caused by inattentive drivers using self-driving cars [2]. Finally, the last article was regarding fatalities due to self-driving vehicles, many of which were due to driver negligence and inattentiveness. However, these fatalities are hindering self-driving technology, as companies are hesitant to advance a technology deemed dangerous [3]. This research has shown that no matter how much the technology improves, consumers still need to take necessary precautions and stay safe for their own sake and to allow for further technological improvements [1,2,3].

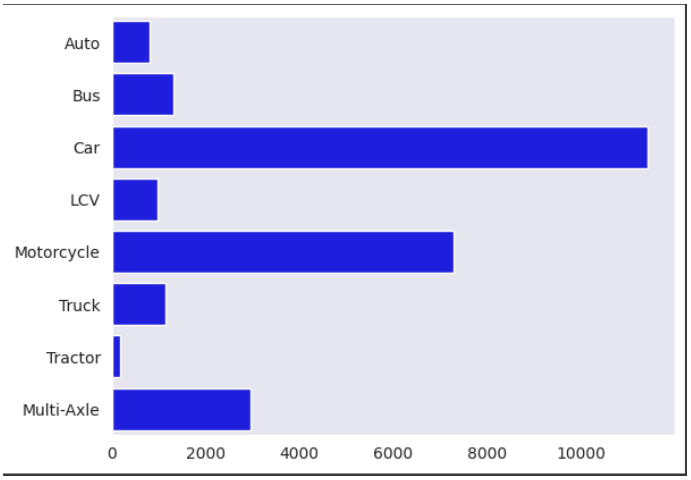


Figure 1. Distribution of vehicle types: shows the distribution of vehicle types present in the dataset

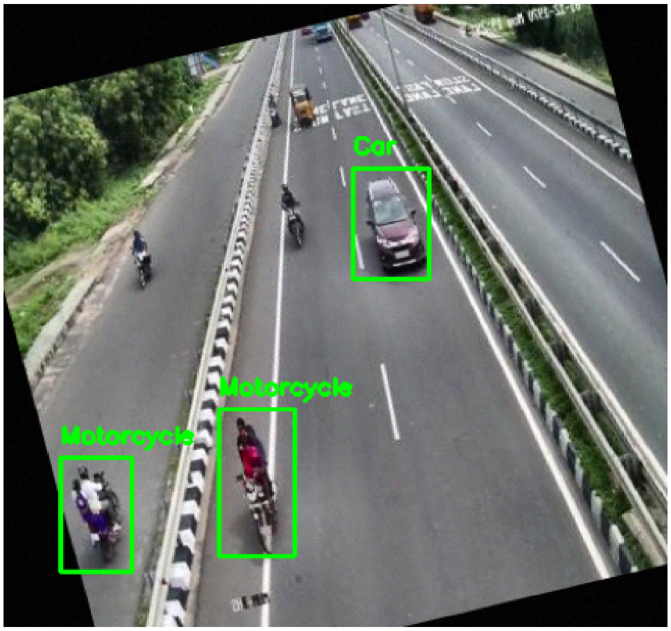


Figure 2. Final preprocessed images: final images once preprocessed with boxes surrounding vehicles with annotations

DATASET

The dataset we used for this project was found on the Kaggle website which is an open-source platform with thousands of datasets primarily for AI use. The dataset, created by Saksham Jain, is titled 'Vehicle Detection 8 Classes | Object Detection' [4]. Given that the research focused on object detection, it was necessary to find a vision dataset; thus, the chosen dataset consists of highway surveillance camera photographs. The dataset contains 8,219 images of vehicles from 8 different classes: autorickshaws (autos), buses, cars, light commercial vehicles (LCV), motorcycles, trucks, tractors, and multi-axle vehicles. Refer to Figure 1 for the proportions of each vehicle type in the dataset. We split the data into 80% for training and 20% for testing our different algorithms. We then began preprocessing all of our training data. Our preprocessing goal was to create a box around each vehicle in each image and place an annotation indicating the vehicle type. This would allow us to train the models to predict the vehicle type based on the annotations. The first step was data augmentation. Since our dataset only had 8,219 images, it was necessary to use data augmentation to artificially increase its size. Data augmentation involves making small changes to existing data to increase the dataset's size [5]. Using data augmentation enabled us to add boxes and annotations for the vehicle type in each image. Adding boxes and annotations around the vehicles allowed our AI models to accurately identify vehicle locations in each picture during training. Once preprocessing was finished, each image in our training dataset resembled the one shown in Figure 2.

METHODOLOGY/MODELS

During our research, one of the main goals was to compare different machine learning models to understand how they can affect various metrics such as precision and accuracy in computer vision. Therefore, we compared three models: VGG16, YOLOV8, and a convolutional neural network (CNN). We chose these models to observe how different architectures would affect our metrics and yield varied results.

VGG16 is a convolutional neural network with 16 layers, each having separate weights. VGG16 has been pre-trained on thousands of images, contributing to its high accuracy. VGG16 comprises thirteen convolutional layers, five max-pooling layers, and three dense layers, summing up to 21 layers, but only sixteen weight layers, hence the name '16' (refer to Figure 3 for a detailed diagram) [6].

YOLOV8, an open-source object detection model, was built by Ultralytics [7]. It consists of three essential blocks: the backbone, neck, and head (refer to Figure 4). The backbone extracts meaningful features from the image input, such as vehicle shapes and sizes. The neck connects the backbone and the head by collecting feature maps from different stages in the backbone and combining them for delivery to the head [8,9]. Feature maps capture essential features from the input image to aid the network in decision-making [10]. Finally, the head generates the final output. It generates bounding boxes for possible objects, assigns confidence scores to each box, and sorts the objects according to their categories. By combining these features, YOLOV8 is both accurate and fast, making it useful for real-time decisions in self-driving vehicles [8,9].

The final model we used was the simplest, a CNN (convolutional neural network). CNNs are the building blocks for most object detection algorithms, including those we previously used. The main purpose of using a CNN was to compare results by controlling more hyperparameters, such as layers and weights, compared to pre-built models like VGG16 and YOLOV8. CNNs comprise three types of layers: convolutional, pooling, and fully-connected layers (refer to Figure 6). Convolutional layers, the building blocks of a CNN, contain input data, a filter, and a feature map. Each image is represented by a 2-dimensional array, and a filter is applied to an area of the image. A dot product is calculated based on the input pixels and the filter. The filter shifts across the image, repeating this action until it has covered the entire input image (refer to Figure 5). The final output is a series of dot products, known as the feature map. Most CNNs have multiple convolutional layers, and after each one, a Rectified Linear Unit (ReLU) transformation is applied to the output feature map. Next, pooling layers, also known as downsampling, perform dimensionality reduction, reducing the number of parameters. Like the convolutional layer, a pooling layer sweeps a filter across the input; however, it does not have weights. Pooling layers reduce complexity, improve efficiency, and limit the risk of overfitting. Finally, the fully-connected layer consolidates information from the previous layers' feature maps to create an output, using the softmax activation function to classify inputs, producing a probability from 0 to 1 [11]. CNNs' basic structure allows for versatility and customization to fit specific problems.

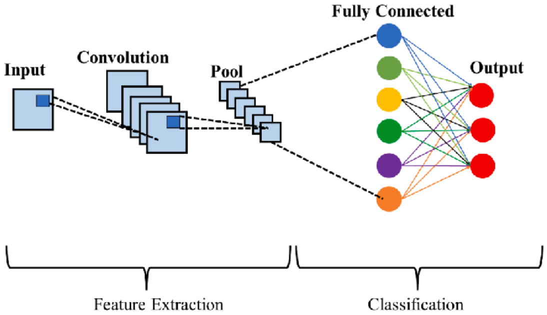


Figure 6. CNN model: shows convolutional neural network architecture and structure

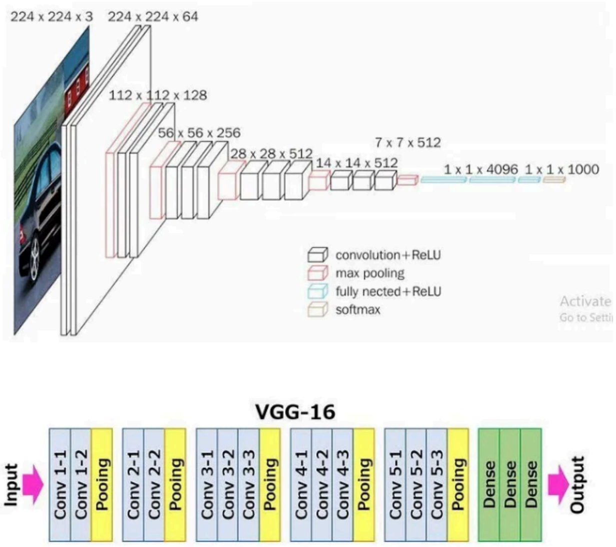


Figure 3. VGG16 model: shows different layers present in VGG16 model and the overall architecture

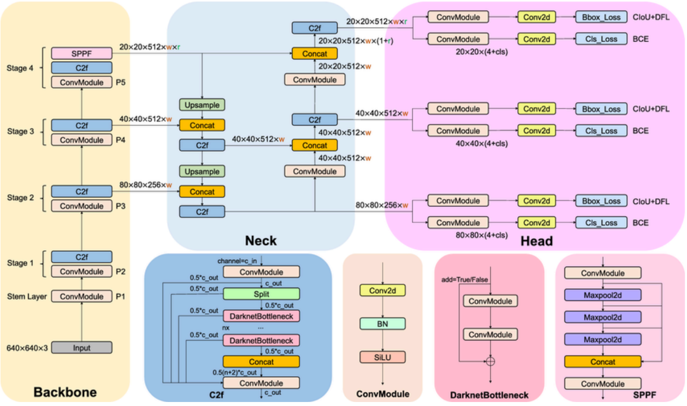


Figure 4. YOLOV8 model: shows different parts of the YOLOV8 model and the overall architecture

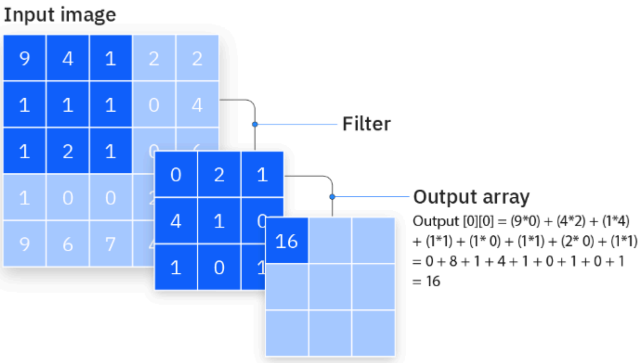


Figure 5. CNN Filter: shows the filtering process in the convolutional and pooling layers and the final dot product

We started with different hyperparameters for each model. For example, our VGG16 model had 50 epochs, while our YOLOV8 and CNN models had only 10 epochs. The VGG16 model had a learning rate of 0.00003, the YOLOV8 model had a default learning rate of 0.01, and the CNN had a learning rate of 0.001. We chose higher epochs and a slower learning rate for VGG16 to update the model's weights with the best ones from each epoch, resulting in a fully optimized model. We specifically used an SSD300 model with a VGG16 backbone and pre-trained weights [14]. We also used an Adam optimizer with a specified learning rate and a learning rate scheduler to reduce the rate when a plateau in validation loss was detected [13]. Our training loop involved an SSD model tracking and updating the learning rate, training, and validation losses. After each epoch, the learning rate scheduler was updated with the validation loss, and if the learning rate changed, the best model weights were reloaded. It also saved the best model weights when a lower validation loss was achieved and logged the losses. Finally, it saved the best model weights to a file.

Since YOLOV8 is designed for object detection and computer vision, we didn’t need as many epochs to achieve good results, and adding more hurt our precision and accuracy [15]. We also used an Adam optimizer for this model to help detect and minimize training losses. For our CNN, we kept the model simple to compare it to the other two models designed for computer vision and object detection. Our CNN had five layers: two convolutional layers, one pooling layer, and two fully-connected layers. We used an Adam optimizer for this model for the same reasons mentioned previously. The first layer takes one input and outputs 32 feature maps, which are processed by the second layer. The second layer outputs 64 feature maps. The pooling layer applies 2x2 max-pooling to the 64 feature maps, reducing their spatial dimensions by half, making them more manageable for subsequent layers. The first fully-connected layer condenses the information into 128 features and feeds it to the final layer, which outputs 9 features corresponding to our 9 vehicle types [16]. The batch size for all our models was 32. Batch size refers to the number of samples fed into the model at each iteration of the training process and must match the architecture of the CPU capabilities [12].

RESULTS AND DISCUSSION

After training and testing all of our models, we evaluated each by running an image through each model and comparing the results with the correct annotations (see Figure 15). We found that VGG16 and YOLOV8 were the best models, while our CNN model performed the worst. Between VGG16 and YOLOV8, VGG16 had the highest metrics. The two metrics we used to evaluate performance were precision and recall. Precision in machine learning is the quality of a positive prediction made by the model and is calculated by the equation shown in Figure 7 [17]. Recall measures how often a machine learning model correctly identifies positive instances (true positives) from all the actual positive samples in the dataset [18]. It is calculated with the equation shown in Figure 8. We chose these two metrics because of the importance of false positives and negatives in object detection. Precision highlights false positives, which means our model falsely classifies an object into one of our nine vehicle classes. This is critical because, in a fully-autonomous vehicle, a false positive could mean the car swerving off the road due to a false detection. Recall highlights false negatives, where the model fails to identify a vehicle, potentially leading to a crash. By using both metrics, we ensured that we considered both false positives and negatives when comparing our models, instead of relying solely on accuracy.

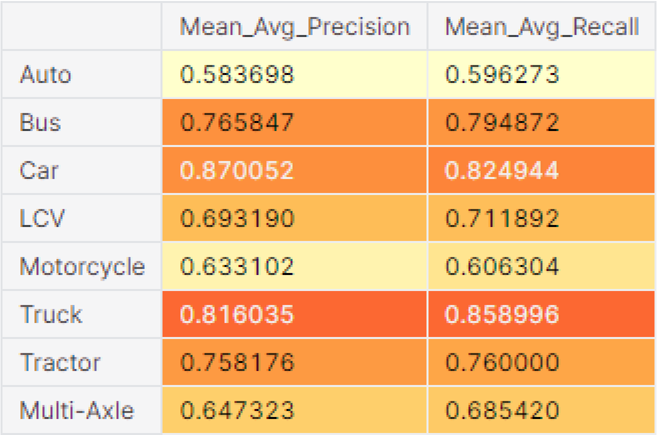
$$Precision = \frac{True\ Positive(TP)}{True\ Positive(TP) + False\ Positive(FP)}$$

Figure 7. Precision Equation: shows the equation for calculating precision in machine learning

$$Recall\ = \frac{True\ Positive(TP)}{True\ Positive(TP) + False\ Negative(FN)}$$

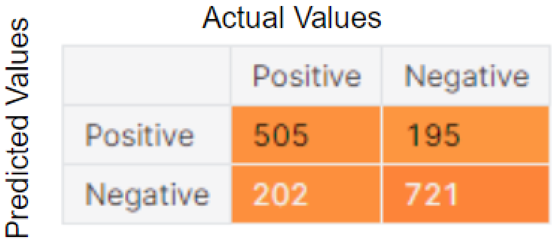
Figure 8. Recall Equation: shows the equation for calculating recall in machine learning

For all our models, we adjusted different hyperparameters to optimize precision and recall, including learning rates, epochs, batch sizes, and different optimizers. For our VGG16 model, we increased the learning rate to 0.0001 from 0.00003 and decreased the number of epochs from 50 to 25. This reduced the training time but significantly lowered our precision and recall metrics. We then increased the epochs to 75 and the learning rate to 0.000001, but this took too long (over a day) and only minimally improved precision and recall. We decided to keep the original hyperparameters: 50 epochs, a learning rate of 0.00003, and a batch size of 32. This setup resulted in a mean precision of 0.7209 and a mean recall of 0.7298, the highest among all three models. Figure 9 shows the precision and recall for each vehicle type in the VGG16 model, and Figure 10 shows the confusion matrix.



	Mean_Avg_Precision	Mean_Avg_Recall
Auto	0.583698	0.596273
Bus	0.765847	0.794872
Car	0.870052	0.824944
LCV	0.693190	0.711892
Motorcycle	0.633102	0.606304
Truck	0.816035	0.858996
Tractor	0.758176	0.760000
Multi-Axle	0.647323	0.685420

Figure 9. VGG16 Table: shows the precision and recall for each vehicle type in testing VGG16 model



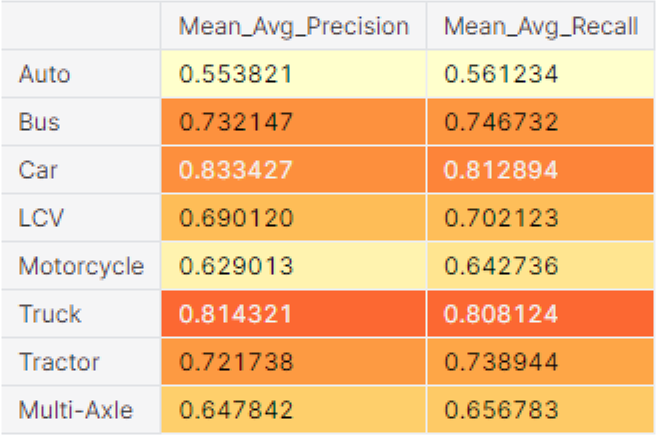
Actual Values

Predicted Values

	Positive	Negative
Positive	505	195
Negative	202	721

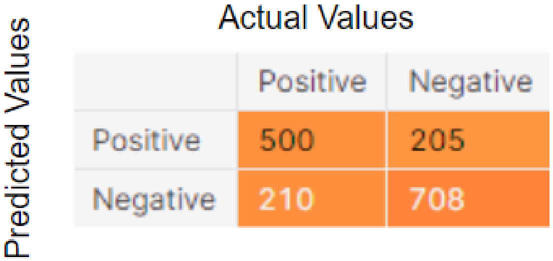
Figure 10. VGG16 Confusion Matrix: shows all true positives/negatives and false positives/negatives while testing VGG16 model

For our YOLOV8 model, we tried increasing and decreasing the epochs to 25 and 5, respectively, but both changes lowered the metrics. Changing the learning rate from 0.01 to 0.001 did not significantly affect metrics but slowed the model, so we kept the original learning rate. We changed the batch size from 16 to 32, which slightly decreased precision and accuracy by 0.0001 percent. Thus, we kept the batch size at 32. The final setup for YOLOV8 was 10 epochs, a learning rate of 0.01, and a batch size of 32, yielding a precision of 0.7028 and a recall of 0.7086, the second-best among the models. Figure 11 shows the precision and recall for each vehicle type in the YOLOV8 model, and Figure 12 shows the confusion matrix.



	Mean_Avg_Precision	Mean_Avg_Recall
Auto	0.553821	0.561234
Bus	0.732147	0.746732
Car	0.833427	0.812894
LCV	0.690120	0.702123
Motorcycle	0.629013	0.642736
Truck	0.814321	0.808124
Tractor	0.721738	0.738944
Multi-Axle	0.647842	0.656783

Figure 11. YOLOV8 Table: shows the precision and recall for each vehicle type in testing YOLOV8 model



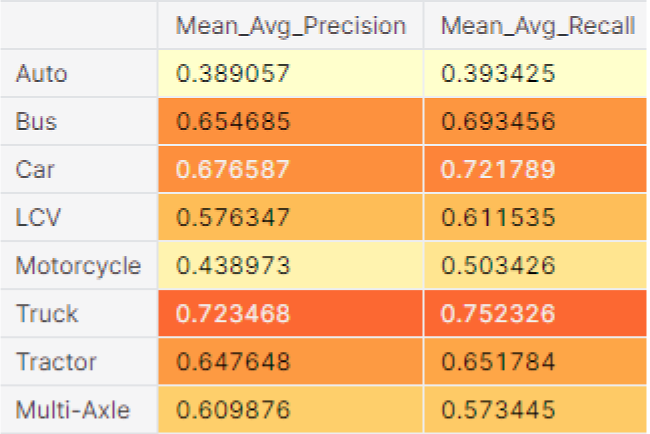
Actual Values

Predicted Values

	Positive	Negative
Positive	500	205
Negative	210	708

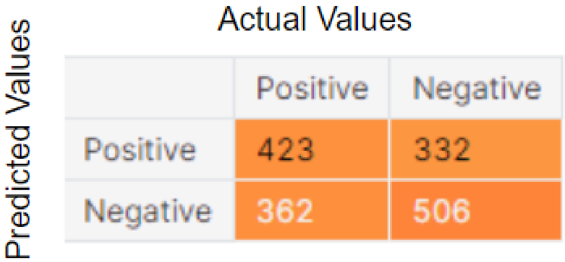
Figure 12. YOLOV8 Confusion Matrix: shows all true positives/negatives and false positives/negatives while testing YOLOV8 model

For our CNN, we experimented with similar and different hyperparameters. Changing the number of epochs, learning rate, and batch size either did not affect or worsened the metrics. Therefore, we kept the original settings: 10 epochs, a learning rate of 0.001, and a batch size of 32. We also experimented with the number of layers. Initially, the CNN had one convolutional layer and one fully-connected layer, but this significantly impacted the metrics. Increasing to two convolutional layers and two fully-connected layers improved results. Adding more layers worsened the metrics, so we kept the configuration with two convolutional layers, one pooling layer, and two fully-connected layers. This setup yielded a precision of 0.5896 and a recall of 0.6126. These metrics were lower than the other two models because VGG16 and YOLOV8 are pre-trained on thousands of images and designed for computer vision, whereas the CNN was created from scratch [19,20]. Figure 13 shows the precision and recall for each vehicle type in the CNN model, and Figure 14 shows the confusion matrix.



	Mean_Avg_Precision	Mean_Avg_Recall
Auto	0.389057	0.393425
Bus	0.654685	0.693456
Car	0.676587	0.721789
LCV	0.576347	0.611535
Motorcycle	0.438973	0.503426
Truck	0.723468	0.752326
Tractor	0.647648	0.651784
Multi-Axle	0.609876	0.573445

Figure 13. CNN Table: shows the precision and recall for each vehicle type in testing CNN model



		Actual Values	
		Positive	Negative
Predicted Values	Positive	423	332
	Negative	362	506

Figure 14. CNN Confusion Matrix: shows all true positives/negatives and false positives/negatives while testing CNN model

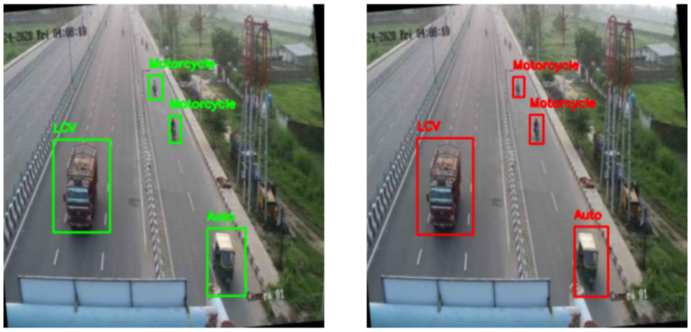


Figure 15. Testing Output: compares the picture with the correct annotations (left) to the outputted picture with annotations by models (right)

Overall, the application, testing, and evaluation of all three models were effective, with only a few flaws. The dataset could have been larger, we could have spent more time hyperparameter tuning, and we could have tested more models. Models for object detection and computer vision usually have tens of thousands of images because high accuracy is crucial for real-world applications. Our dataset had only 8,219 images split into training and testing sets. A larger dataset might have improved our metrics, possibly reaching the high 0.8s and 0.9s. Our hyperparameter tuning was limited by time constraints, preventing us from testing different values in small increments. More detailed tuning might have yielded better results. Finally, testing more models could have identified one performing better than VGG16.

CONCLUSION

In summary, our main question during this research was how object detection can improve the decision-making of self-driving cars. Our goals for this research were to try out different models and observe their effects on precision and recall, understand the factors that impact these metrics, and gain a deeper understanding of object detection models and their application in self-driving vehicles. During our research, we saw how different object detection models could detect vehicles on a highway and classify them into nine separate vehicle types. We analyzed different metrics and compared these models, finding that VGG16 outperformed YOLOV8 and CNN. We also tested various hyperparameters and their effects on these metrics.

We gained a deeper understanding of object detection models through the processes of data pre-processing, training, and testing different models. We learned what contributes to the success of these models and what causes them to perform worse than expected. We believe that our model performed well considering the constraints we had and that our research was overall successful. With more data, more time for hyperparameter tuning, and the ability to test additional models, we would likely have achieved even better results. However, we achieved reasonably high metrics on all models and accomplished a lot.

Regarding the next steps, we aim to try more types of machine learning models to see if this could lead to further improvements. Additionally, we plan to obtain more data, as object detection models require extensive datasets for optimal performance, and our current dataset of 8,219 images is insufficient.

In conclusion, our research went very well, and we accomplished all our goals. We are satisfied with the results and hope to develop our research further in the future.

ACKNOWLEDGMENTS

I want to personally thank Hassan Azmat and InspiritAI for providing me with the resources and tutelage to conduct my research. Their guidance and help allowed me to understand the different AI models I used and taught me how to code them. They also helped me understand the different parts needed to create a research paper and oversaw the construction of it to guide me in the right direction.

REFERENCES

[1] Gupta, A. (2021, February 23). *Deep learning for object detection and scene perception in self-driving cars: Survey, Challenges, and open issues*. Science Direct. https://www.sciencedirect.com/science/article/pii/S2590005621000059

[2] Cummings, M. L. (2023, July 30). *What self-driving cars tell us about AI risks*. IEEE Spectrum. https://spectrum.ieee.org/self-driving-cars-2662494269

[3] McArdle, M. (2023, May 31). *Opinion | self-driving cars will make roads safer — but not until humans bow out*. The Washington Post. https://www.washingtonpost.com/opinions/2023/05/31/self-driving-cars-safety-drivers/

[4] Jain, S. (2020, December 18). *Vehicle detection 8 classes: Object detection*. Kaggle. https://www.kaggle.com/datasets/sakshamjn/vehicle-detection-8-classes-object-detection

[5] *What is data augmentation? - data augmentation techniques explained* - AWS. (n.d.). https://aws.amazon.com/what-is/data-augmentation/

[6] Learning, G. (2021, September 23). *Everything you need to know about VGG16*. Medium. https://medium.com/@mygreatlearning/everything-you-need-to-know-about-vgg16-7315defb5918

[7] “A New State-of-the-Art Computer Vision Model.” YOLOv8, yolov8.com/.

[8] Torres, J. (2024, January 15). *Yolov8 architecture: A deep dive into its architecture*. YOLOv8. https://yolov8.org/yolov8-architecture/

[9] Pedro, J. (2023, December 11). *Detailed explanation of yolov8 architecture-part 1*. Medium. https://medium.com/@juanpedro.bc22/detailed-explanation-of-yolov8-architecture-part-1-6da9296b954e

[10] *What is a feature map?*. Educative. (n.d.). https://www.educative.io/answers/what-is-a-feature-map#:~:text=Feature%20maps%20are%20a%20core,data%20and%20retaining%20important%20information

[11] *What are convolutional neural networks?*. IBM. (2021, October 6). https://www.ibm.com/topics/convolutional-neural-networks

[12] Brownlee, J. (2022, August 15). *Difference between a batch and an epoch in a neural network*. MachineLearningMastery.com. https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/#:~:text=The%20batch%20size%20is%20a,passes%20through%20the%20training%20dataset

[13] Vishwakarma, N. (2024, April 26). *What is Adam Optimizer?*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2023/09/what-is-adam-optimizer/#:~:text=The%20Adam%20optimizer%2C%20short%20for,Stochastic%20Gradient%20Descent%20with%20momentum

[14] *Ssd300_vgg16*. ssd300_vgg16 - Torchvision main documentation. (n.d.). https://pytorch.org/vision/main/models/generated/torchvision.models.detection.ssd300_vgg16.html

[15] Ultralytics. (2024, June 10). *Train*. Train - Ultralytics YOLO Docs. https://docs.ultralytics.com/modes/train/#augmentation-settings-and-hyperparameters

[16] Kalita, D. (2024, April 19). *Basics of CNN in Deep Learning*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2022/03/basics-of-cnn-in-deep-learning/#:~:text=The%20four%20key%20components%20of,and%20classification%20in%20image%20data

[17] *Precision*. C3 AI. (2023, September 19). https://c3.ai/glossary/machine-learning/precision/#:~:text=Precision%20is%20one%20indicator%20of,the%20number%20of%20false%20positives

[18] *Accuracy vs. precision vs. recall in machine learning: What’s the difference?* Evidently AI - Open-Source ML Monitoring and Observability. (n.d.). https://www.evidentlyai.com/classification-metrics/accuracy-precision-recall#:~:text=Recall%20is%20a%20metric%20that,the%20number%20of%20positive%20instances

[19] GeeksforGeeks. (2024, March 21). *VGG-16: CNN model*. https://www.geeksforgeeks.org/vgg-16-cnn-model/

[20] Husnain, A. (2023, October 8). *Understanding Yolov8: A revolution in object detection*. Medium. https://medium.com/@ali.hxnyn13/understanding-yolov8-a-revolution-in-object-detection-550c91a98aad