Self-driving Car Problem

Tran Quoc Truong* (truongtqse173295@fpt.edu.vn)[a,1], **and Tang Quang Hieu (hieutq10@fe.edu.vn)**[b,2]

[a]Student, University; [b]Mentor, University

Professor: Responsible for the experiment

Abstract*—The main challenges of self-driving cars are detecting traffic signs, semantically segmenting lanes, and developing algorithms to control the car. In our study, we will approach the basic ways to build a self-driving car. We use Unity, a game development platform, to simulate the virtual environment, including shadows, snow, stop lines, lane marking, and various objects such as rocks, leaves, and obstacles like other cars on the road. Along the roadside, it has grass, trees, and traffic signs. The method we use to detect traffic signs is color filters to crop the traffic sign image and the YOLOv8 Classifier model to classify, and the data we use is provided by VIA dataset v1.0 including over 18k images of 7 classes: stop, left, right, straight, no_left, no_right, and unknown. Secondly, to semantic segment the lane, we choose YOLOv8 Segmentation, which is fast enough to run in real-time, and the data we use is provided by the VIA dataset v1.0 including approximately 8k images and 1,5k self-labeled images from that virtual environment, using augmentation to increase to 3k images. In the controller phase, we use the DIP algorithm to steer the car into the middle of the lane and rely on the traffic sign when the lane is available. And the result is the car finished the simulation in time without any trouble. In conclusion, we can build your self-driving car with only basic methods and algorithms.*

Keywords*—Mathematics for AI, Neural Networks, Self-driving Car...*

1. Introduction

Autonomous vehicles, in general, and in particular self-driving cars, can "sense" their surroundings and operate independently without any human intervention. In terms of technology, self-driving cars apply the most advanced models such as edge computing, IoT, and especially Artificial Intelligence (AI), thereby not only stopping at collecting data with sensors. , self-driving cars can also observe and learn from situations, make their own decisions when participating in traffic, or even predict actions and risks. Data needs to be processed on-site to make immediate decisions, while still ensuring safety and accuracy. Regardless of Internet connection, optimal autonomous vehicles can also effectively communicate with other vehicles to collect more complete and detailed real-time environmental information.

The Society of Automotive Engineers (SAE) currently defines 6 levels of driving automation ranging from Level 0 (fully manual) to Level 5 (fully autonomous). These levels have been adopted by the U.S. Department of Transportation. [5]

Levels of driving automation:

- Level 0 - No automation: Manual control. The human performs all driving tasks (steering, acceleration, braking, etc, ...).
- Level 1 - Driver assistance: The vehicle features a single automated system (e.g it monitors speed through cruise control).
- Level 2 - Partial automation: ADAS. The vehicle can perform steering and acceleration. The human still monitors all tasks and can take control at any time.
- Level 3 - Conditional automation: Environmental detection capabilities. The vehicle can perform most driving tasks, but human override is still required.
- Level 4 - High automation: The vehicle performs all driving tasks under specific circumstances. Geofencing is required. Human override is still an option.
- Level 5 - Full automation: The vehicle performs all driving tasks under all conditions. Zero human attention or interaction is required.

Autonomous cars rely on sensors, actuators, complex algorithms, machine learning systems, and powerful processors to execute software. Autonomous cars create and maintain a map of their surroundings based on a variety of sensors situated in different parts of the vehicle. Radar sensors monitor the position of nearby vehicles. Video cameras detect traffic lights, read road signs, track other vehicles, and look for pedestrians. Lidar (light detection and ranging) sensors bounce pulses of light off the car's surroundings to measure distances, detect road edges, and identify lane markings. Ultrasonic sensors in the wheels detect curbs and other vehicles when parking. Sophisticated software then processes all this sensory input, plots a path, and sends instructions to the car's actuators, which control acceleration, braking, and steering. Hard-coded rules, obstacle avoidance algorithms, predictive modeling, and object recognition help the software follow traffic rules and navigate obstacles.

But in our study, we will approach the basic ways to build a self-driving car. We use Unity, a game development platform, to simulate the virtual environment, including shadows, snow, stop lines, lane marking, and various objects such as rocks, leaves, and obstacles like other cars on the road. Along the roadside, it has grass, trees, and traffic signs. The method we use to detect traffic signs is color filters to crop the traffic sign image and the YOLOv8 Classifier model to classify, and the data we use is provided by VIA dataset v1.0 including over 18k images of 7 classes: stop, left, right, straight, no_left, no_right, and unknown. Secondly, to semantic segment the lane, we choose YOLOv8 Segmentation, which is fast enough to run in real-time, and the data we use is provided by the VIA dataset v1.0 including approximately 8k images and 1,5k self-labeled images from that virtual environment, using augmentation to increase to 3k of images. In the controller phase, we use the DIP algorithm to steer the car into the middle of the lane and rely on the traffic sign when the lane is available.

2. Methodology

2.1. Basic Theoretical Background.

1. **Derivative** in mathematics, is the rate of change of a function concerning a variable. Derivatives are fundamental to the solution of problems in calculus and differential equations.

$$f'(x_0) = \lim_{x \to x_0} \frac{f(x) - f(x_0)}{x - x_0} \tag{1}$$

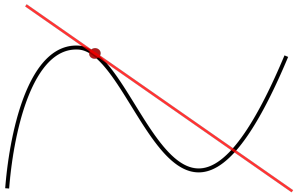

Figure 1. *Example of Derivative*

2. **Gradient Descent** is an optimization algorithm for finding a local minimum of a differentiable function. Gradient descent in machine learning is simply used to find the values of a function's parameters (coefficients) that minimize a cost function as far as possible.
Gradient Descent Algorithm:
   (a) **Initialize Parameters**: Start with initial values for the parameters (weights) of the model.
   (b) **Compute Gradient**: Calculate the gradient (1) of the cost function concerning each parameter.
   (c) **Update Parameters**: Adjust the parameters in the opposite direction of the gradient to minimize the cost function, using a fraction of the gradient scaled by a learning rate.
   (d) **Repeat**: Iterate steps 2 and 3 until convergence criteria are met, such as a maximum number of iterations or when the change in the cost function becomes negligible.

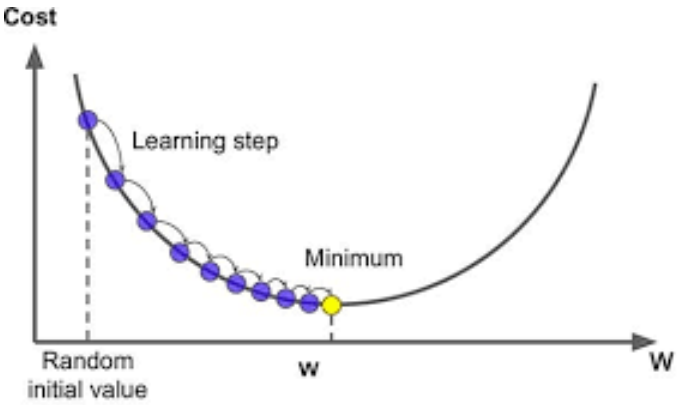

Figure 2. *Example of Gradient Descent*

3. **The Chain Rule** In differential calculus, the chain rule is a formula used to find the derivative of a composite function. If **y** is a function of **u** (**y** = $f(u)$), and **u** is a function of **x** (**u** = $f(x)$), then the derivative of **y** with respect to **x** can be found by multiplying the derivative of **y** with respect to **u** by the derivative of **u** with respect to **x**.

$$\frac{dy}{dx} = \frac{dy}{du} * \frac{du}{dx} \tag{2}$$

4. **Activation Functions** An Activation Function decides whether a neuron should be activated or not. This means that it will decide whether the neuron's input to the network is important or not in the process of prediction using simpler mathematical operations.
   (a) **Why we should use Activation Functions:**
      i. Create the non-linear for the model.
      ii. Keep the output values within a certain range.
   (b) **Some popular Activation Functions:**
      i. Sigmoid

$$f(x) = \frac{1}{1 + e^{-x}} \tag{3}$$

      ii. Tanh (Hyperbolic Tangent)

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{4}$$

      iii. ReLu(Rectified Linear Unit)

$$f(x) = \max(0, x) \tag{5}$$

      iv. Softmax (commonly used in the output layer)

$$f_i(x) = \frac{e^{x_i}}{\sum_{j=1}^{C} e^{x_j}} \tag{6}$$

5. **PID algorithm** The PID algorithm, or Proportional-Integral-Derivative controller, is a control algorithm widely used in engineering and automation. It continuously adjusts control inputs based on the difference between a desired setpoint and a measured process variable.

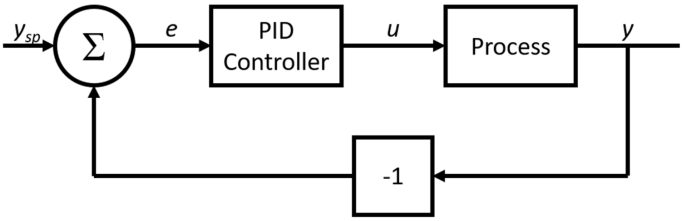

Figure 3. *PID Diagram*

PID controllers consist of three terms:

   (a) Proportional (K_p): Reacts proportionally to the current error signal.
   (b) Integral (L_i): Integrates the error signal over time to eliminate steady-state error.
   (c) Derivative (K_d): Dampens system response by reacting to the rate of change of the error signal.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\, d\tau + K_d \frac{de(t)}{dt} \tag{7}$$

2.2. Advanced Theoretical Background.

1. **Neural Network** A neural network is a machine learning program, or model, that makes decisions like the human brain, by using processes that mimic the way biological neurons work together to identify phenomena, weigh options, and arrive at conclusions. [7]

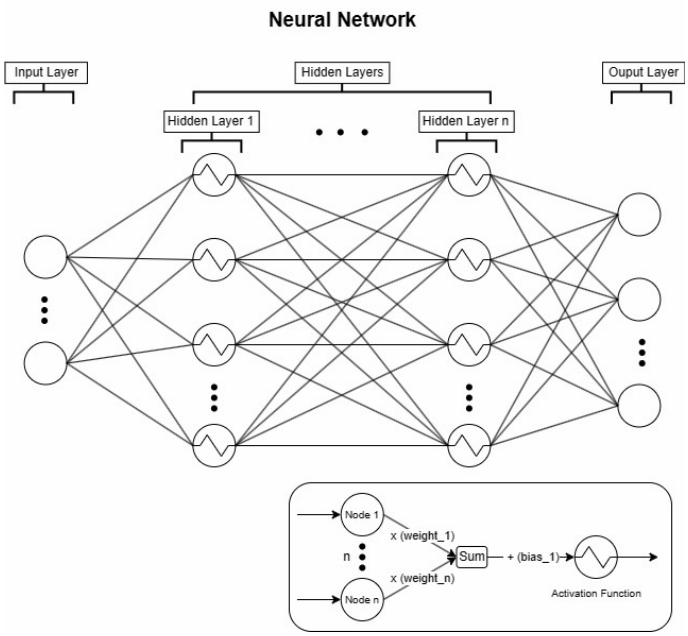

Figure 4. *Example of Neural Network*

How Neural Network work!

(a) **Training**

Step 1: Set the input/output data into the input/output layer.

Step 2: Determine how many hidden layers and their nodes are.

Step 3: Initialize random w_i (weight value) for each node, commonly using Normal Distribution (Gaussian Distribution) to pick value.

Step 4: Initialize random b_i (weight value) for each node, commonly equal to 0.

Step 5: Use Back-Propagation (Chain rule 2) to fit the model w_i, b_i.

(b) **Prediction**: Use forward propagation to calculate the output values relying on w (weight) and b (bias) belonging to each node, which are pretrained.

2. **Convolutional Layers** are a key part of Convolutional Neural Networks (CNNs), which are widely used in computer vision and image processing. These layers perform a mathematical operation called convolution on their input.
In convolution, a small filter (also called a kernel or filter, commonly (3x3) or (5x5)) is moved across the input data. At each position, the filter multiplies its values elementwise with the corresponding values in the input and sums them up to produce a single value in the output, creating what's called a feature map.

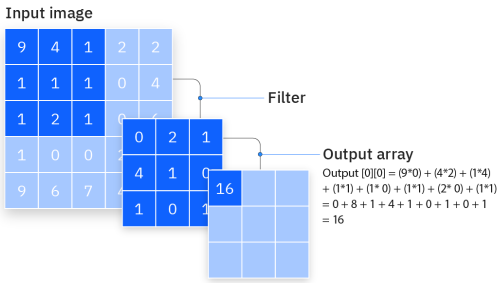

Figure 5. *Example of Convolution*

3. **DeConvolution** is totally opposite process from Convolution. It is also known as **Transposed Convolution**. In DeConvolution, the Feature map gets converted into an Image. Converting Convoluted image into Original Image is DeConvolution.
Convolution adds up the information spread in various pixels to one pixel, whereas DeConvolution spreads the information present in one pixel to various pixels.

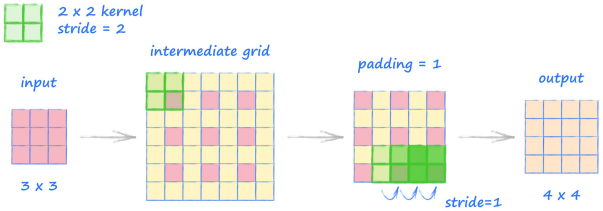

Figure 6. *Example of Deconvolution*

4. **Pooling Layers** are an essential component of Convolutional Neural Networks (CNNs). These layers are used to reduce the size of the input data while keeping important information...
Specifically, pooling layers typically perform a simple aggregation operation such as taking the maximum value (max pooling) or computing the average value (average pooling) from small regions of the feature map generated by convolutional layers.

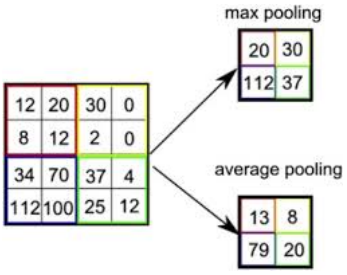

Figure 7. *Example of Pooling*

5. **Unpooling layers**, the opposed to pooling layers, are the process of undoing the pooling operation, where the original spatial dimensions of the data are restored, typically in order to recover the spatial information lost during pooling.

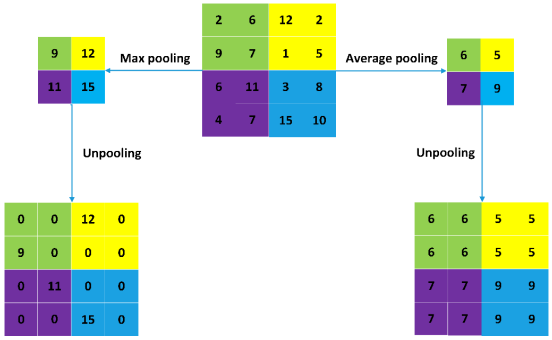

Figure 8. *Example of Unpooling*

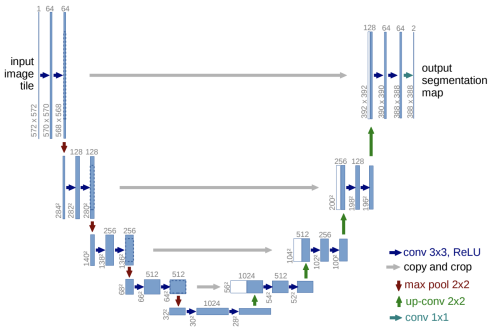

Fig. 1. U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

Figure 10. *U-net*

2.3. Deep Learning Models Architecture.

1. **CNN (Convolution Neural Network)** A convolutional neural network (CNN) is a category of machine learning model, a type of deep learning algorithm well suited to analyzing visual data. CNNs, sometimes called convents, use principles from linear algebra, particularly convolution operations, to extract features and identify patterns within images. Although CNNs are predominantly used to process images, they can also be adapted for audio and other signal data.
- In an image classifier, CNN can split into two sets of layers, the first set is extracting features, including convolutional layers, and pooling layers (which will be introduced later on), and the last set is fully connected layers, which are a traditional neural network. [2]

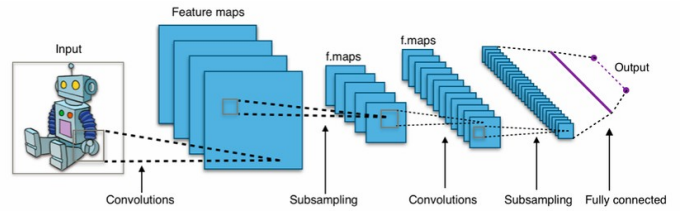

Figure 9. *Example of CNN*

(a) **Extracting features**:

- Convolutional Layers.
- Pooling Layers.

(b) **Fully Connected Layers**: refers to the traditional Neural Network (1). In these layers, the inputs consist of extracted feature layers, which are flattened, into 1 dimension, and the outputs are the classified layers.

2. **U-Net** is a convolutional neural network that was developed for biomedical image segmentation at the Computer Science Department of the University of Freiburg. The network is based on a fully convolutional neural network whose architecture was modified and extended to work with fewer training images and to yield more precise segmentation. Segmentation of a 512×512 image takes less than a second on a modern GPU... [3]

It consists of a contracting path and an expansive path:

- The contracting path follows the typical architecture of a convolutional network. It consists of the repeated application of two 3x3 convolutions (unpadded convolutions), each followed by a rectified linear unit (ReLU) and a 2x2 max pooling operation with stride 2 for downsampling. At each downsampling step, we double the number of feature channels.
- Every step in the expansive path consists of an upsampling of the feature map followed by a 2x2 convolution (“up-convolution”) that halves the number of feature channels, a concatenation with the correspondingly cropped feature map from the contracting path, and two 3x3 convolutions, each followed by a ReLU. The cropping is necessary due to the loss of border pixels in every convolution.

At the final layer, a 1x1 convolution is used to map each 64-component feature vector to the desired number of classes. In total, the network has 23 convolutional layers.

3. **Faster R-CNN** is an object detection model that improves on **Fast R-CNN** [1] by utilising a region proposal network (RPN) [4] with the CNN model [2].

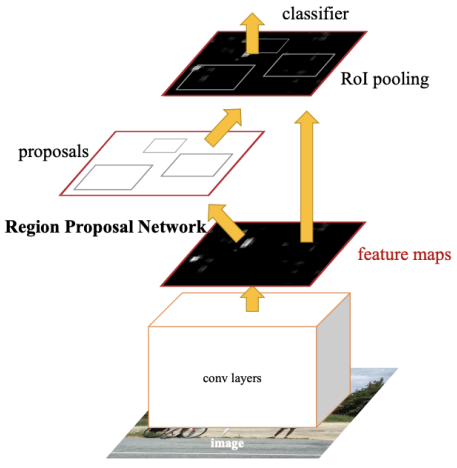

Figure 11. *Faster R-CNN is a single, unified network for object detection. The RPN module serves as the ‘attention’ of this unified network.*

The RPN shares full-image convolutional features with the detection network, enabling nearly cost-free region proposals. It is a fully convolutional network that simultaneously predicts object bounds and objectness scores at each position. The RPN is trained end-to-end to generate high-quality region proposals, which are used by Fast R-CNN for detection. RPN and Fast R-CNN are merged into a single network by sharing their convolutional features: the RPN component tells the unified network where to look.

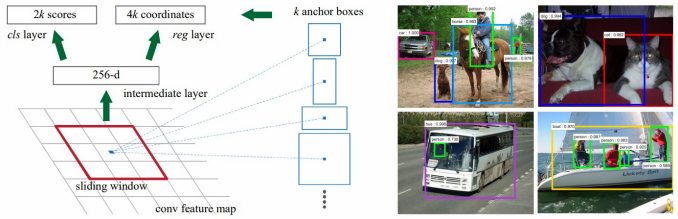

Figure 12. *Left: Region Proposal Network (RPN). Right: Example detections using RPN proposals on PASCAL VOC 2007 test. Our method detects objects in a wide range of scales and aspect ratios.*

4. **YOLOv8** YOLOv8 is the latest version of the YOLO object detection model. This latest version has the same architecture as its predecessors 6 but it introduces numerous improvements compared to the earlier versions of YOLO such as a new neural network architecture that utilizes both Feature Pyramid Network (FPN) and Path Aggregation Network (PAN) and a new labeling tool that simplifies the annotation process. This labeling tool contains several useful features like auto labeling, labeling shortcuts, and customizable hotkeys. The combination of these features makes it easier to annotate images for training the model [8].
YOLOv8 uses an anchor-free model with a decoupled head to independently process objectness, classification, and regression tasks. This design allows each branch to focus on its task and improves the model's overall accuracy. In the output layer of YOLOv8, they used the sigmoid function as the activation function for the objectness score, representing the probability that the bounding box contains an object. It uses the softmax function for the class probabilities, representing the objects' probabilities belonging to each possible class. [9]
YOLOv8 uses CIoU [76] and DFL [114] loss functions for bounding box loss and binary cross-entropy for classification loss. These losses have improved object detection performance, particularly when dealing with smaller objects.
YOLOv8 also provides a semantic segmentation model called YOLOv8-Seg model. The backbone is a CSPDarknet53 feature extractor, followed by a C2f module instead of the traditional YOLO neck architecture. The C2f module is followed by two segmentation heads, which learn to predict the semantic segmentation masks for the input image. The model has similar detection heads to YOLOv8, consisting of five detection modules and a prediction layer. The YOLOv8-Seg model has achieved state-of-the-art results on various object detection and semantic segmentation benchmarks while maintaining high speed and efficiency.
YOLOv8 can be run from the command line interface (CLI), or it can also be installed as a PIP package. In addition, it comes with multiple integrations for labeling, training, and deploying.
Evaluated on MS COCO dataset test-dev 2017, YOLOv8x achieved an AP of 53.9% with an image size of 640 pixels (compared to 50.7% of YOLOv5 on the same input size) with a speed of 280 FPS on an NVIDIA A100 and TensorRT.

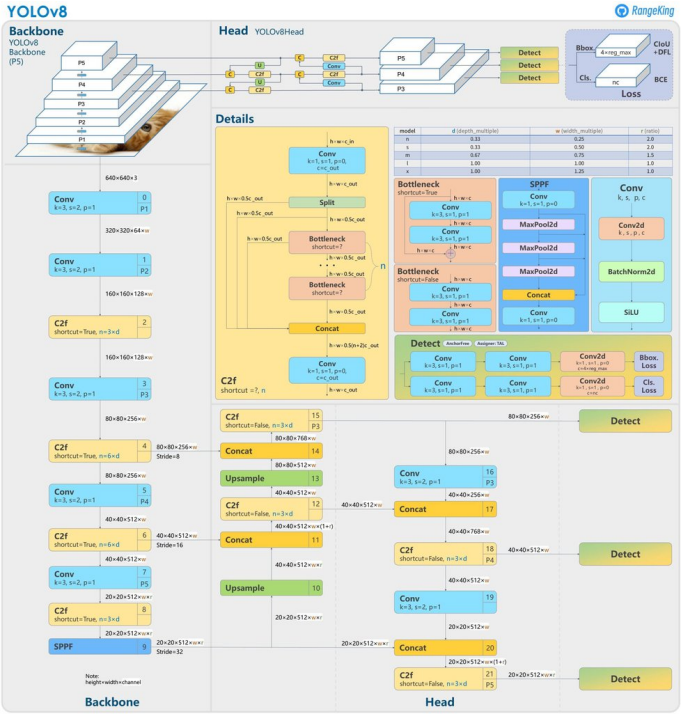

Figure 13. *YOLOv8 Architecture. The architecture uses a modified CSPDarknet53 backbone. The C2f module replaces the CSPLayer used in YOLOv5. A spatial pyramid pooling fast (SPPF) layer accelerates computation by pooling features into a fixed-size map. Each convolution has batch normalization and SiLU activation. The head is decoupled to process objectness, classification, and regression tasks independently*

2.4. Traffic Signs Detection. Traffic sign detection is a crucial aspect of self-driving cars as it provides the algorithm with the information necessary to follow road safety rules. In this phase, we use VIA dataset v1.0 including over 18k images of 7 classes: stop, left, right, straight, no_left, no_right, and unknown. To detect the signs, we use the filter color to cut the images, which may contain the traffic signs, and use the YOLOv8 Classifier model to classify them.

1. Traffic Sign Cropping Algorithm:

Step 1: Filter the images by the color of the sign.

Step 2: Get bounding boxes of the sign from the mask (the filtered images).

Step 3: Filter the bounding boxes to remove those that are too small, too big, or have a height-to-width ratio that is too large.

Step 4: Merge the intersecting bounding boxes to reduce them to a single bounding box for the model.

Step 5: Repeat step 3 to get the final bounding boxes.

2. Traffic Sign Classification:

   (a) Data Preprocessing:
   - Data validation: Despite of provided from VIA dataset v1.0, we have to check the image from each class to make sure that the image is in the correct class.
   - Data splitting: we split data into 2 sets, training set (80%) and validation set (20%).
   - Data Augmentation: the augmentation methods we use to increase images in the training set are:
     + Brightness: ±10%.
     + Zoom: ±10%.
     + Rotation: ±15∘.
     + Width shift: ±20%.
     + Height shift: ±20%.
     + Channel shift: ±50%.

   (b) Setup Model:
   - Model: YOLOv8 Classifier.
   - Image size: 32px.
   - Batch size: 64.
   - Epoch: 100.

   (c) Training: After setting up the model and augmenting data, we conduct training.

2.5. Lane Segmentation. Lane segmentation is a crucial part of autonomous vehicles, enabling them to perceive lanes and navigate within them similar to traffic signs detection. In this phase, we use the dataset provided by VIA dataset v1.0 including approximately 8k images and 1,5k self-labeled images. To segment the lane, we use YOLOv8 Segmentation, which is fast enough to run in real time.

1. Data Preprocessing:
   - Data labeling: the data we get from the Car Simulate Environment, and we use a pre-trained Segment Anything Model (SAM) to label data. After that, we check and label manually.
   - Data validation: Despite of provided from VIA dataset v1.0, we have to check the image from each labeled in the right ways.
   - Data splitting: we split data into 2 sets, training set (80%) and validation set (20%).
   - Data Augmentation: the augmentation methods we use to increase images in the training set are:
     + Brightness: ±10%.
     + Flip: Horizontal
     + Zoom: ±10%.
     + Rotation: ±15∘.
2. Setup Model:
   - Model: YOLOv8 Segmentation.
   - Image size: 640px.
   - Batch size: 64.
   - Epoch: 100.
3. Training: After setting up the model and augmenting data, we conduct training.

2.6. Self-driving Cars Controller Algorithms. Self-driving Cars Controller Algorithms with Lane Segmentation and Traffic Signs Detection are the foundation of Self-driving cars, which make decisions about what have to do like turn left when seeing a left sign or stop when seeing a stop sign. Self-driving Cars Controller Algorithms take the Lane Segmentation phase and Traffic Signs Detection phase outputs as input and use the algorithm to steer and accelerate.

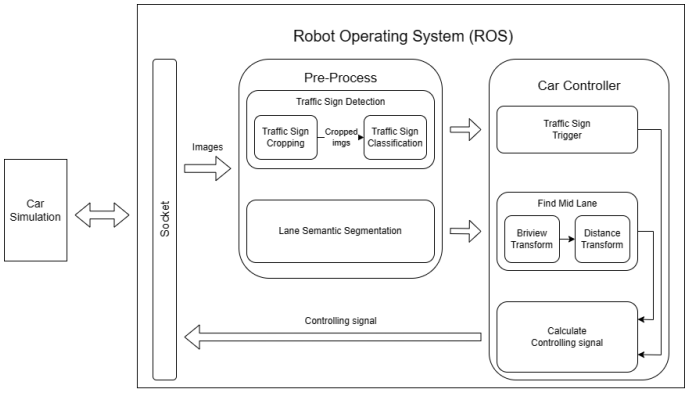

Figure 14. *Workflow of Self-driving Car*

Algorithm 1 Self-Driving Car Algorithm

function GET_MIDPOINT_LANE($mask$)
  $birdview\_mask \leftarrow birdview\_transform(mask)$
  $trans\_mask \leftarrow distance\_transform(birdview\_mask)$
  $mid\_point \leftarrow get\_midpoint(trans\_mask)$
  return mid_point
end function

function MAIN()
  $segmented\_mask \leftarrow lane\_segment()$
  $midpoint \leftarrow get\_midpoint\_lane(segmented\_mask)$

  $last\_signal \leftarrow straight$
  $curr\_signal, conf\_signal \leftarrow traffic\_sign\_detect()$

  if $last\_signal \neq curr\_signal$ **then**
    if $conf\_signal \geq 0.95$ **then**
      $last\_signal \leftarrow curr\_signal$
    end if
  end if

  if $last\_signal = straight$ **then**
    Go straight #Use midpoint and PID() algorithm [6]
  else if $last\_signal = left$ and $is\_left\_lane()$ **then**
    Turn Left
    $last\_signal \leftarrow straight$
  else if $last\_signal = right$ and $is\_right\_lane()$ **then**
    Turn Right
    $last\_signal \leftarrow straight$
  else if $last\_signal = stop$ and $is\_stop\_line()$ **then**
    Stop
    $last\_signal \leftarrow straight$
  end if
end function
$main()$

3. Results and Discussion

3.1. Traffic Signs Detection.

1. Traffic Sign Cropping: After setting up the algorithm, this algorithm can perform well and crop the image containing the traffic sign. After that, the algorithm can crop the traffic sign perfectly. However, the drawback of the Traffic Sign Cropping Algorithm is that the algorithm crops the traffic sign by color, so it may not perform well in any environment.

2. Traffic Sign Classifier: During the training phase, we use various metrics to measure how well the model performed. Based on the confusion matrix, which is normalized, we can realize that the model is well-performed in all classes except the right_class which has lower accuracy ($\approx$ 91%) than other classes, which are higher 97%. In the right class, the model is highly confused with the left_class. Therefore, to improve this problem, our approach involves data of right_class but also left_class to enhance the whole model.

Figure 15. *Confusion matrix normalized.*

In the evaluation model process, we observe that the algorithm can reduce the loss function smoothly after 50 iterations including the training and validation phase. The classifier model performs with high accuracy ($\approx$ 98%) with top 1 and keeps it stable after 75 iterations. So that, we can know that the classifier model is learning from the data very well. As mentioned above, we can improve the classifier model's accuracy by collecting more data in left_class and right_class, or simpler augmentation data of both classes.

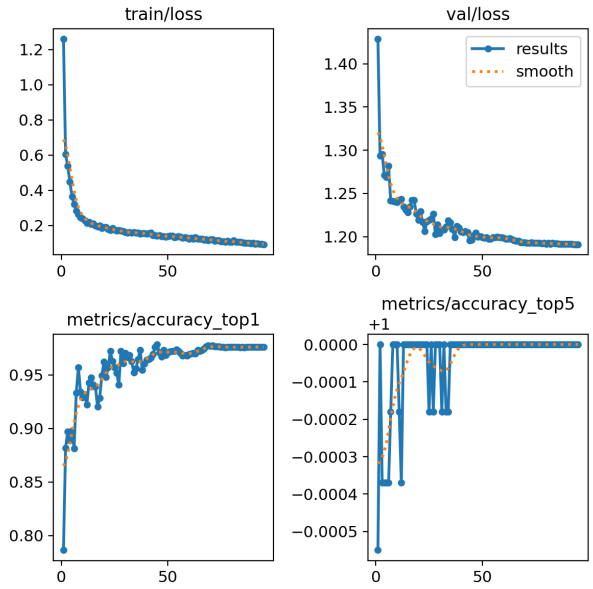

Figure 16. *Result of the Classifier Model.*

This is the result of the Traffic Sign Detection Algorithm step-by-step

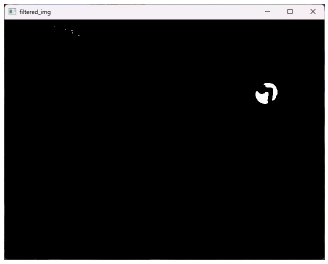

Figure 17. *Traffic Sign Cropping Step 1.*

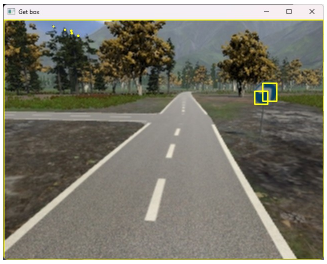

Figure 19. *Traffic Sign Cropping Step 2.*

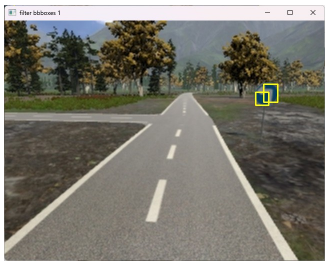

Figure 18. *Traffic Sign Cropping Step 3.*

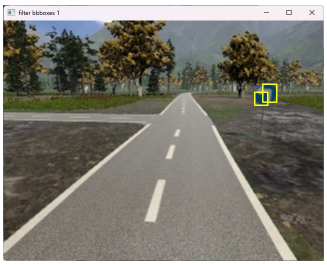

Figure 20. *Traffic Sign Cropping Step 4.*

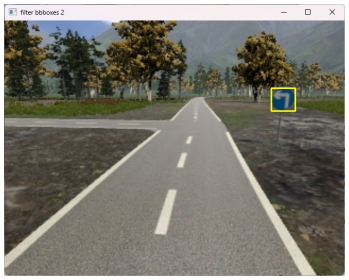

Figure 21. *Traffic Sign Cropping Step 5.*

3.2. Lane Segmentation. With a large of data including provided and self-labeled, the segmentation model can perform its task correctly. The segmentation model gives high accuracy in the segmentation lanes ($\approx$ 97%) and the background (100%). So that, our self-driving car can "sense" the lane correctly.

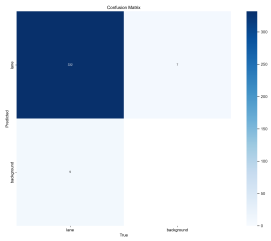

Figure 22. *Confusion matrix.*

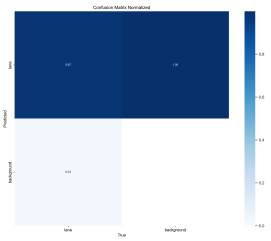

Figure 23. *Confusion matrix normalized.*

The provided results can give us the information that the loss function in the training phase, in general, significantly reduced after nearly 100 iterations, and that proves that the 100 epochs may not be the most effective but optimal. In the validation phase, the loss function's fluctuations are more intense but trending is going down through iterations. After that results, we can state that the segmentation model is well-learned based on the provided data.

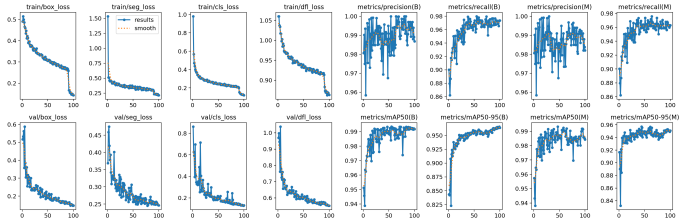

Figure 24. *Result of the Segmentation Model.*

This is a result of the Lane Segmentation.

Figure 25. *Original Image.*

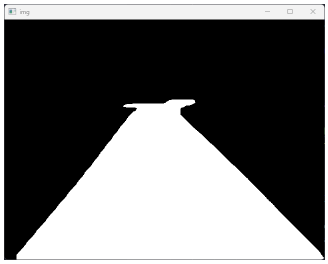

Figure 26. *Segmented Image.*

3.3. Ability to Self-driving. Because the self-driving algorithm is only tested in a simulated environment. Using Unity, a game development platform, to simulate the virtual environment, including shadows, snow, stop lines, lane marking, and various objects such as rocks, leaves, and obstacles like other cars on the road. Along the roadside, it has grass, trees, and traffic signs. Because completing the Traffic Signs Detection task and the Lane Segmentation well, the self-driving algorithm can perform very well, and it finished the race in 125.8s, with full marks (5/5). Despite of completed well, we must admit that the algorithms of the self-driving car project are primitive and simple, and all of that leads to self-driving cars only performing in some specific environments.

4. Conclusion

In the process of building self-driving cars, we have witnessed remarkable advancements in technology and computer science, which start from basic theories, advanced theories and the last is deep learning models. However, this self-driving car version can only perform in some specific environments. To improve our project, we can collect more data and add more sensors to have more information about objects or human surroundings. Overall, your research contributes to a deep insight into the self-driving car problem that helps you build one of your own.

References

[1] R. Girshick, "Fast r-cnn", *Computer Vision and Pattern Recognition*, Sep. 2015, [Accessed: April 9, 2024]. [Online]. Available: https://arxiv.org/abs/1504.08083.

[2] K. O'Shea and R. Nash, "An introduction to convolutional neural networks", *Neural and Evolutionary Computing*, Dec. 2015, [Accessed: April 9, 2024]. [Online]. Available: https://arxiv.org/abs/1511.08458.

[3] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation", *Computer Vision and Pattern Recognition*, May 2015, [Accessed: April 9, 2024]. [Online]. Available: https://arxiv.org/abs/1505.04597.

[4] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks", *Computer Vision and Pattern Recognition*, Jan. 2016, [Accessed: April 9, 2024]. [Online]. Available: https://arxiv.org/abs/1506.01497.

[5] SAE International. "Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles". (2018), [Online]. Available: https://www.sae.org/standards/content/j3016_202104/ (visited on 04/30/2021).

[6] H. Razali, M. M. Kamal, and N. A. Razak, "Vehicle speed control for autonomous vehicle using pid controller", *Autonomous*, Nov. 2019, [Accessed: April 9, 2024]. DOI: 10.35940/ijrte.D5179.118419. [Online]. Available: https://www.researchgate.net/publication/364089330_Vehicle_Speed_Control_for_Autonomous_Vehicle_using_PID_Controller.

[7] B. Mehlig, "Machine learning with neural networks", *Machine Learning*, Oct. 2021, [Accessed: April 9, 2024]. [Online]. Available: https://arxiv.org/abs/1901.05639.

[8] D. Reis, J. Kupec, J. Hong, and A. Daoudi, "Real-time flying object detection with yolov8", *Computer Vision and Pattern Recognition*, May 2023, [Accessed: April 9, 2024]. [Online]. Available: https://arxiv.org/abs/2305.09972.

[9] J. Terven and D. Cordova-Esparza, "A comprehensive review of yolo architectures in computer vision: From yolov1 to yolov8 and yolo-nas", *Computer Vision and Pattern Recognition*, Feb. 2024, [Accessed: April 9, 2024]. [Online]. Available: https://arxiv.org/abs/2304.00501v7.