LiDAR and Camera Fusion in Autonomous Vehicles

by

Jie (Jenny) Zhang

Submitted in partial fulfillment of the requirements

for the degree of Master of Applied Science

at

Dalhousie University

Halifax, Nova Scotia

August 2022

DEDICATION PAGE

This thesis is dedicated to my loving daughter.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

LiDAR and camera can be an excellent complement to the advantages in an autonomous vehicle system. Various fusion methods have been developed for sensor fusion. Due to information lost, the autonomous driving system cannot navigate complex driving scenarios. When integrating the camera and LiDAR data, to account for loss of some detail of characters when using late fusion, we could choose a convolution neural network to fuse the features. However, the current sensor fusion method has low efficiency for the actual self-driving task due to the complex scenarios. To improve the efficiency and effectiveness of context fusion in high density traffic, we propose a new fusion method and architecture to combine the multi-model information after extracting the features from the LiDAR and camera. This new method is able to pay extra attention to features we want by allocating the weight during the feature extractor level.

LIST OF ABBREVIATIONS USED

FGS	Faculty of Graduate Studies
Dal	Dalhousie University
AI	Artificial Intelligence
CNN	Convolutional Neural Network
DC	Driving Score
RC	Route Completion
IM	Infraction Multiplier
ROI	Region of Interest Pooling
SAE	Society of Automotive Engineers
ABS	Anti-lock Braking System
ESP	Electronic Stability Program
ACC	Adaptive Cruise Control
LKA	Lane Keeping Assistance
VW	Volkswagen
BMW	Bayerische Motoren Werke AG
GM	General Motors
BEV	Bird's Eye View
FOV	Front of View
FPS	Frame per Second
FLOPs	Floating Point Operations
SENet	Squeeze and Excitation Network
GRU	Gated Recurrent Units
GPT	Generative Pre-trained Transformer
GPS	Global Position System
GPU	Graphics Processing Unit
FLOPs	Floating Point Operations
MSE	Mean Square Error
MLP	Multilayer Perceptrons
RNN	Recurrent Neural Networks

LRN	Local Response Normalization
PID	Proportional Integral Derivative
PCA	Principal Component Analysis
ReLU	Rectified Linear Units
Radar	Radio Detection and Ranging
RGB	Red Green Blue
LiDAR	Light Detection and Ranging

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Dr. Jason Gu, for his guidance and patience. For the past two years, he has guided me on my academic path and helped me develop my research skills to complete my thesis successfully. I really appreciate the opportunities he has provided for me to grow professionally.

I would like to thank the committee members, Dr. Kamal El-Sankary and Dr. Srinivas Sampalli. For their valuable comments and suggestions, and for their feedback, which made this thesis more complete.

I would like to thank my classmates, Zhengyang Wang, Ming Yan, Hanxiang Zhang, and Yang Ge, for their help studying and doing experiments. We discussed and helped each other; it will be a cherished memory in my life.

I would like to thank the staff in the Electrical and Computer Engineering Department for their support, especially Ms. Tamara Cantrill, Departmental Administrator.

Finally, I would like to give special thanks to my husband and my families for their great patience and invaluable support!

CHAPTER 1 INTRODUCTION

The following sections outline the background for the research work, including research motivation, overviews of autonomous vehicles, sensor fusion, deep-learning network, and research contributions.

Research Motivation: Artificial intelligence (AI) advances have pushed the development of self-driving vehicles (AVs). Driven by extensive data from various sensing devices and advanced computing resources, AI has become essential for self-driving vehicles to sense their surroundings and make appropriate decisions while in motion. More attention is given to environmental perception tasks in autonomous driving to achieve the goal of full automation. Therefore, it is extremely important to improve the effectiveness and accuracy of data fusion. This paper proposes a new deep learning based artificial intelligence algorithm model to fuse 3-D point cloud and front-view RGB image inputs on a feature level to enhance the efficiency and robustness of data fusion between LiDAR and camera sensors.

Macroscopically: There are several ways to achieve intelligent driving. The most common way is to improve the accuracy performance of the subtasks' results through, for instance, 3-D object detection and segmentation. Our paper is focused on the end-to-end driving method, which inputs the raw data from the sensors' feed through the Convolution Neural Network and directly mapping to the vehicle control signals. There is comparatively less end-to-end research than on the centralized subtask model of previous studies in academia. It is gaining popularity due to the low requirements for sensor data and the reduced requirements for data annotation [7]. While most approaches for end-to-end driving rely on a single input modality, autonomous driving systems are often outfitted with cameras and LiDAR sensors [2]. Our paper proposes the end-to-end multimodal method, which fuses the LiDAR and front-view-camera data.

Microcosmically: To design the deep learning network structure, we created and implemented five enhanced versions of Transfuser [8] with different fusion methods.

The main ideas for designing CNN architecture in this thesis include:

1. Proposing to add the Squeeze-and-Excitation Network into the fusion module, which mainly learns the correlation between channels, filters the attention against channels, performs feature extraction on the feature layer in the channel dimension and multimodally fuses the selected information.

2. Using the BiFusion method to fuse LiDAR and image information [9]

3. Trying different attention mechanisms to combine the features at the feature extraction level. For example, we tried to use cross-attention instead of simple concatenation.

4. Using the Convnext network instead of ResNet for feature extraction since the article [10] proved the powerful performance of the Convnext model for image recognition.

5. Trying to adjust the size of the pooling layer to solve the problem of the convolutional position feature information loss.

1.1. Autonomous Vehicle

A self-driving car, also known as a driverless car, computer-driven car, crewless car, or self-driving car, is a vehicle that requires no driver assistance. As an automated vehicle, a self-driving car can sense its environment and navigate without needing a human operator [4][5].

As indicated in Fig. 1, the autonomous vehicle system may be separated into four primary groups. Various sensors on the vehicle are used to sense the surrounding environment. These are hardware components that collect environmental data. Next, the perception block processes sensor data into relevant information. The planning subsystem employs the output of the perception block for both short- and long-range path planning. The control module ensures that the vehicle follows the route determined by the planning subsystem and provides the vehicle control orders [3].

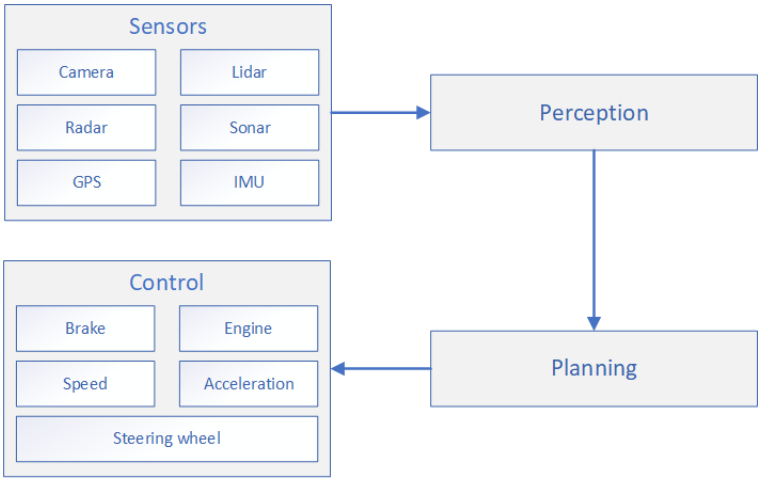


Figure 1 Block diagram of the autonomous vehicle system [3]

The development of automated driving technology is a steady and iterative process rather than a significant shift from zero to one. The most command-used standard for grading autonomous driving systems is that set by the Society of Automotive Engineers (SAE). The standards developed by different organizations can vary slightly, but the basic concept is the same.

The article [1] summarizes the six levels from Level 0 (manual driving) to Level 5 (fully autonomous driving).

Level 1: The anti-lock braking system (ABS) and the electronic stability program (ESP), now standard in cars, belong to the L1 level. In addition, fixed speed cruise control, adaptive cruise control (ACC), and lane-keeping assist (LKA) also belong to the L1 class because they can only steer the vehicle in one direction (lateral or longitudinal).

Level 2: When LKA and ACC are used together, the automobile reaches the L2 level. The vehicle driver must keep an eye on their surroundings and be prepared to take control at any moment for systems in L2 and below. This is critical and is the main reason why many L2 level vehicles have traffic accidents (i.e., the driver expects too much from the system and does not always keep an eye on the surroundings while driving). A vehicle is at the L3 level if it has some kind of pilot system, such Traffic Jam Pilot.

Level 3: Level 3 means that in certain specific scenarios (e.g., highways or traffic jams ~~etc.~~), the driver may release their hands, feet, and eyes while operating the car and only take control when the system signals them to do so, eliminating the need for constant attention to the state of the road. The driver has switched to becoming a passenger in this instance.

Level 4: An L4 level system only exists in demo vehicles. Such manufacturer's vehicles only achieve a couple hours of automatic driving on the road without artificial takeover. The biggest difference from L3 is that there is no artificial takeover; in a limited scenario L4 can realize the vehicle's fully autonomous driving.

Level 5: The limited scene restriction will also be removed at Level 5. There is no steering wheel, everyone is a passenger, and whole control of the vehicle belongs to the system.

For different levels of autonomous driving and different application scenarios, the configuration options for sensors are different. In order to achieve fully autonomous driving, sensors are a very important part of the process. We need to use the information collected by the existing sensors to accurately sense the situation around the vehicle (such as pedestrians, cars and other obstacles) and to efficiently and accurately use the information collected by each sensor to make judgments about the car's execution actions.

Perception, decision-making, and control are the three main elements of an autonomous driving system. These three modules roughly represent the eyes, brain, and limbs of a biological system. The decision system (brain) decides the next action to be taken based on the environment and the set target, and the control system (limbs) executes these actions, such as steering, accelerating, and braking. The perception system (eyes) is responsible for understanding the surrounding obstacles and road information.

The perception system has two functions: environment perception and vehicle localization. Environmental perception is responsible for identifying different moving and stationary objects (e.g., cars, people, buildings~~, etc.~~) and collecting various road information (e.g., drivable areas, lane lines, traffic signs, traffic lights), which involves the usage of numerous

sensors (e.g., cameras, LiDAR, millimetre-wave radar). Vehicle placement is dependent on environmental awareness information to establish the location of the vehicle in the environment, which needs high-precision maps, inertial navigation (IMU), and global positioning system support (GPS) [6].

1.2. Sensor Fusion

Autonomous cars may include cameras, LiDAR, radar, sonar, a global positioning system (GPS), an inertial measurement unit (IMU), and wheel odometry as sensors. Sensors in automotive vehicles gather data that is evaluated by the autonomous car's computer and utilized to drive the vehicle.

Camera: The cameras are a popular device for observing the environment. Photo sensible surfaces (picture planes) detect light emitted from the surroundings through a camera lens (placed in front of the sensor) to provide crisp images of the surroundings [2].

LiDAR: The LiDAR is an abbreviation for Light Detection and Ranging (LiDAR). The LiDAR measures the distance between the sensor and a nearby object by using an infrared laser beam. Some LiDARs employ longer wavelengths, which operate better in rain and fog. However, the majority of modern LiDARs emit light with a 900-nm wavelength [3].

Radar: Radar is an abbreviation for Radio Detection and Ranging. Long-range radar is a seventy-seven GHz microwave radar with limited resolution, but it can measure velocity and identify vehicles and obstructions up to two hundred meters away. In the twenty-four GHz and seventy-six GHz frequency bands, short or medium-range radar is an affordable and established technology. This sensor can detect the velocity and distance, but its precision is limited by its wide beams and long wavelengths, and its return signals are complicated [3].

Advantages and Disadvantages**:**

Table 1 Advantages, disadvantages and max working distances of Radar, camera, LiDAR and Ultrasonic sensor

Type	Advantages	Disadvantages	Maximum working distance
MMW-Radar	1) Extended travel distance 2) Accessible for radial speed 3) Applicable in all conditions	1) Impractical for static items 2) Easily generating false alarms	5m-200m
Camera	1) Excellent discernibility 2) Available lateral velocity 3) Able to be distributed in color	1) Extensive computational burden 2) Light scattering 3) Vulnerable to weather conditions 4) Unavailable for radial velocity	250m (depending on the lens)
LiDAR	1) Wide field of view (FOV) 2) High range resolution 3) High angle resolution	1) Unbearable under poor weather 2) High cost	200m
Ultrasonic	1) Inexpensive	1) Low pixel density 2) Not suited for high velocity	2m

Due to the different sensors having different characteristics, each with advantages and disadvantages, we can combine the advantage of different types of sensors to adapt to complex condition (weather, daytime, and nighttime). For example: The camera is the most commonly used sensor in perception systems, with the advantage of being able to extract rich texture and color information; therefore, it is suitable for the target classification task. However, the disadvantage is that they have a weak perception of distance and are highly influenced by lighting conditions.

The LiDAR is appropriate for medium-to-close range target identification and ranging since it correctly senses the distance and form of objects, relatively compensating for the limitations of cameras. However, the disadvantages include higher cost, difficulty in mass production,

limited sensing distance, and being more affected by the weather.

The millimeter-wave radar is appropriate for low-cost sensing systems or used as a supplemental with other sensors due to its all-weather operation, increased accuracy in measuring target speed and distance, larger sensing range, and comparatively low cost. However, the disadvantage is the low resolution in height and laterality, and the limited ability to sense stationary objects.

In order to combine the pros and cons of each sensor, an efficient and accurate multimode sensor fusion method is a key research point in the study.

Sensor Fuse Method: According to the fusion state, the fusion methods can be divided into early, middle and late fusion depending on where they appear. The methods of multi-sensor information fusion can be decided by what to fuse as data-level fusion, feature-level fusion and decision-level fusion [4].

Early fusion (Data level): Fig 2 shows that the collected raw data are fused directly from the sensors with early fusion [4].

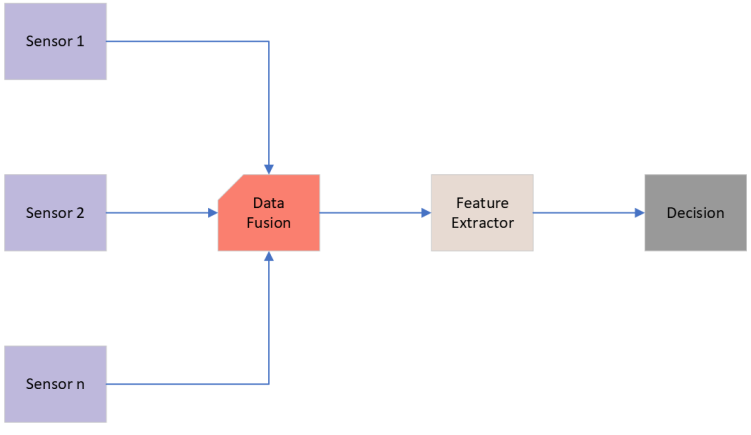


Figure 2 Early fusion operation flow

The network learns the joint characteristics of many modalities at an early stage. The advantage of early fusion is it maximizes the use of the raw data information. However, early

fusion has low processing and memory requirements as it analyses several sense modalities concurrently. This occurs at the expense of model rigidity. When, for instance, an input is replaced with a new sensing modality or the number of input channels is increased, the previously fused network must be entirely retrained. Early fusion is susceptible to spatial-temporal data misalignment among sensors, which is caused by calibration error, varying sampling rate, and sensor failure.

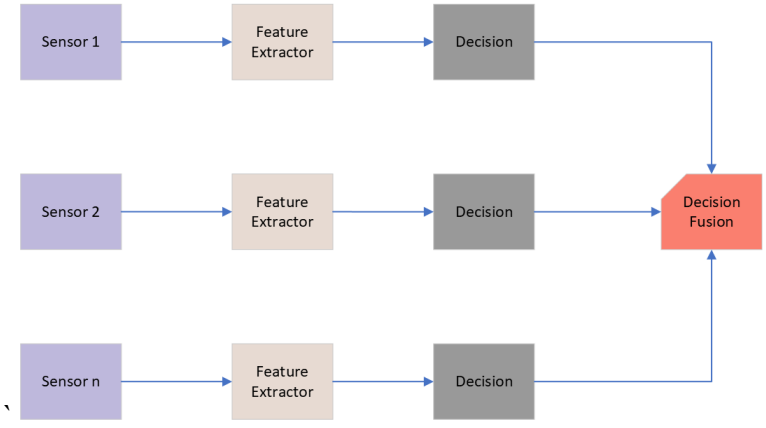


Figure 3 Late fusion operation flow

Late fusion (decision-level): Fig. 3 shows late fusion is at the decision level, which is very adaptable and modular. When a novel sensory modality is introduced, only its domain-specific network requires training, with no impact on existing networks. However, it has a high cost of computation and memory needs. In addition, it discards rich intermediate characteristics that may be highly advantageous when combined [4].

Middle fusion (feature-level): Middle fusion (Fig. 4) happens at the feature level, which has a balance between early and late fusion. It integrates, at intermediate levels, the feature representations from many sensory modalities. This allows the network to learn cross-modalities with diverse feature representations at varying depths.

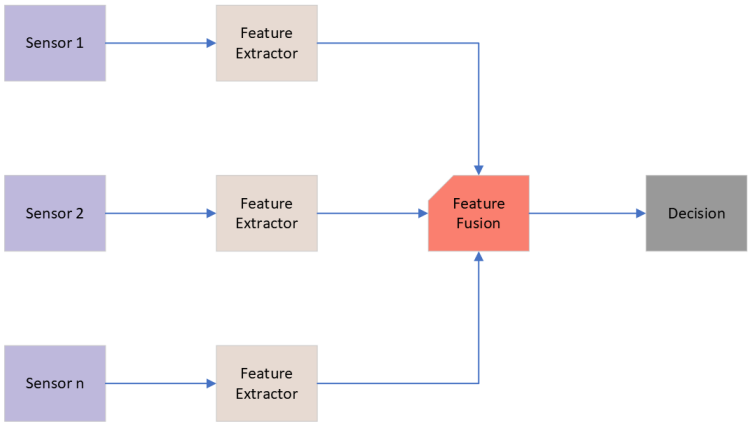


Figure 4 Mid-fusion operation flow

As early and late fusion can suppress intra- or inter-modal interactions, research now focuses on intermediate fusion methods, allowing these fusion operations to be placed in multiple layers of a deep learning model.

After understanding sensors and fusion, we need to know the current status of the application of these sensors in the production or demonstration vehicles on the market. The companies that specialize on autonomous vehicles can be broadly split into two groups. One category is traditional car companies (e.g., VW, BMW, GM, Toyota), new energy car companies (e.g., Tesla) and Tier1 (e.g., Bosch, Continental, Amphoe). The primary goal of these companies is mass production, and they generally focus on L2-level solutions but are currently expanding to L3 level as well [11][12]. The other category encompasses a number of solution providers or startup companies (e.g., Waymo, Momenta, Mobileye, Pony.AI). These companies are developing L4 level self-driving technologies for such uses as Robot-taxi, Robot-truck, and Robot bus [11].

For different levels of autonomous driving and different application scenarios, the sensor configuration options are different. First is the pure vision solution introduced by Tesla. Although Tesla is frequently the first car that comes to mind when discussing autonomous driving, it is only an L2 level system since the driver must always be prepared to take control

of the vehicle. If we compare the system side-by-side at the L2 level, then Tesla's solution is still very competitive [12].

Figure 5 Tesla advanced sensor coverage [13]

As Fig. 5 shows that Tesla's system uses only vision sensors and includes cameras mounted in different locations on the body with multiple focal lengths and fields of view. We observe that these cameras have some redundancy and a three-hundred-sixty-degree field of vision. Thus, Tesla's fusion solution is mainly achieved through deep learning-based multi-camera fusion algorithms [13].

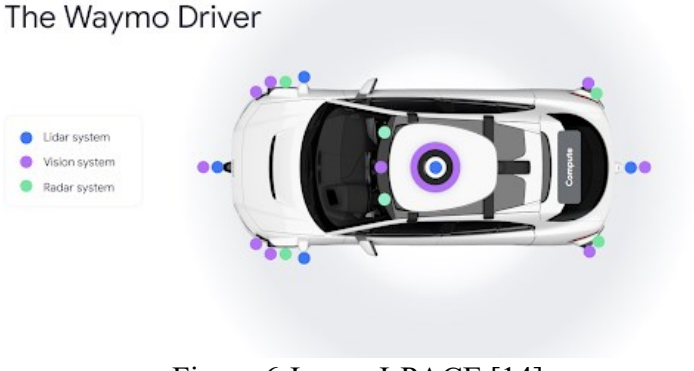


Figure 6 Jaguar I-PACE [14]

Another type of sensor solution, such as the one provided by Waymo, a subsidiary of Google, adds rearward and roof-mounted 360-degree LiDAR in addition to forward-facing LiDAR. The number of LiDAR beams is significantly increased to reach a sensing range of about 300 meters. Therefore, Waymo company is mainly based on deep learning to achieve autonomous

driving by fusing cameras and LiDAR sensors [14].

1.3. Deep-learning Network

Convolutional Neural Network

Basic concepts of convolutional neural networks: The convolution neural network is based on the traditional artificial neural network, which is like the traditional fully connected neural network. The convolutional neural network converts the input image data into two-dimensional matrix format data. Then each layer of the network processes the data in a two-dimensional matrix.

This data processing is suitable for digital images in a 2D matrix format. It can extract the feature values from the image data faster and better than the traditional artificial neural network. It is a powerful class of neural grids designed to process image data.

Nowadays, convolutional neural networks are the basis for almost all academic competitions and commercial applications related to image recognition, target detection, or semantic segmentation. The design of modern convolutional neural networks has benefited from biology, group theory, and a series of complementary experiments. A convolutional neural grid requires fewer parameters than a fully connected grid. Hence, convolution can be easily computed in parallel using a GPU. The convolutional neural networks can also be computed efficiently by sampling to obtain accurate models. Over time, the practitioners have increasingly used convolutional neural networks. Even for one-dimensional sequence structure tasks that typically use a recurrent neural grid (e.g., voice and sound), the use of a convolutional neural network has been increasing by making some clever adjustments to the convolutional network [15].

CNN Architecture:

The basic structure of a convolutional neural network consists of the following components: Input layer, Convolution layer, Pooling layer, Full-connection layer and Output layer [19].

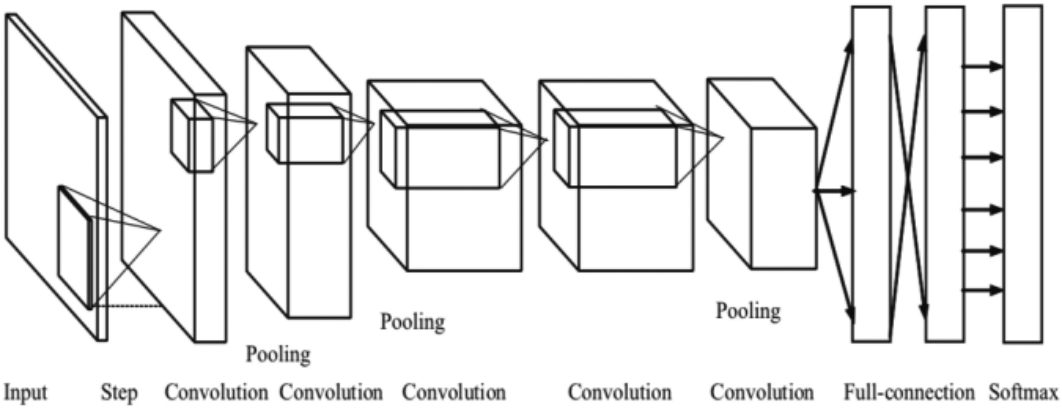


Figure 7 Convolutional neural network architecture [19]

Input Layer:[18] The key task in the input layer is processing input data. For instance, for the image information pre-processing operations are first performed, and the common ways of pre-processing images in the input layer the RGB images are decomposed and divided into three channels of red (R) green (G) and blue (B), where each value is between zero and two hundred fifty-five. The following operations are performed:

- De-meaning: centering each dimension of the input image information to 0 and pulling the center of the sample back to the origin of the coordinate system.

- Normalization: Normalizing the magnitude information to the same range to reduce the interference caused by the difference in the range of values taken by each dimensional data, generally between -1 to 1.

Convolution layer: The data convolute operations are performed in the convolutional layer. The main purpose of convolutional operations is to enhance the feature information of the original data and reduce noise [19]. The definition of 2D convolution as shown in Eq. (1)

$$f(x,y) * g(x,y) = \int_{\tau_1=-\infty}^{\infty} \int_{\tau_2=-\infty}^{\infty} f(\tau_1,\tau_2) \cdot g(x-\tau_1, y-\tau_2) d\tau_1 d\tau_2 \qquad (1)$$

There are couple basic terminologies we need to know first as shown below [20].

Depth: The depth refers to the depth of the graph and the depth of its control output unit, also expressed as the number of neurons connecting the same area. Generally, the number of the convolutional kernels (filters) is the number of the neuron's depth.

Stride: The stride is used to describe the step length of convolutional kernel movement

Zero-padding: Zero-padding is for padding the edges of the image by adding zeros to the edges of the image, thus controlling the size of the output unit space.

Convolution kernel: The convolution kernel refers to a filter where each pixel in the output image is a weighted average of the pixels in a small region of the input image. It is a function of weight. There can be more than one convolution kernel, and the convolution kernel parameters can be trained by error backpropagation.

The convolution operation has three steps:

1. Dot product: As in Fig. 8 [16], each element in the $5 \times 5 \times 3$ blue area is the input layer. After filling in the zero to become the $7 \times 7 \times 3$ input matrix (zero-padding) is multiplied by its corresponding position of the weights (numbers in the filter $w0$), and then added together. The resulting value is used as the $5 \times 5 \times 3$ output matrix for the first element.

2. Sliding window: As in Fig. 8, move the $3 \times 3 \times 3$ weight matrix two-cell to the right (stride is 2).

3. Repeat the operation: Similarly, multiply each element in the dark region with the corresponding weight value and add them together. The resulting value is used as the second element of the output matrix; repeat the above "dot product-sliding window" operation until all values of the output matrix are obtained. The convolution kernel "slides" over the 2-dimensional input data, multiplies the elements of the current input part of the matrix, and then converges the result to a single output pixel value, repeating this process until the whole image is traversed. This process is called convolution, the weight matrix is the convolution kernel, and the image after the convolution operation is

called the feature map [19].

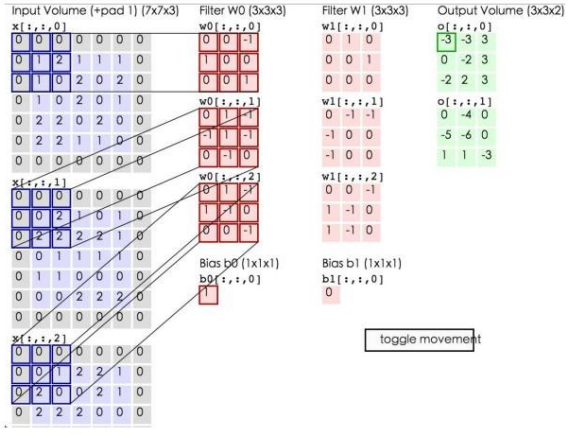


Figure 8 Convolution calculate [16]

Fig. 8 shows two channels (output volume) generated by two convolution kernels (filters) of 3×3×3 convolution on RGB images [16].

Pooling layer: The convolution layer is often followed by a downsampling layer, whose main purpose is to reduce the length and width of the matrix and to reduce the parameters of the input matrix. Calculating the average or maximum value of a particular feature over a region of the image, this aggregation operation is called pooling [16].

The pooling operation is to extract features and reduce the amount of data passed to the next stage. It is very important role of the pooling layer. So, image processing is compressing the image while preserving the key feature information in the image. In other words, after each convolution operation of the original image, the size of the image is reduced by downsampling operation in Fig. 9 [21].

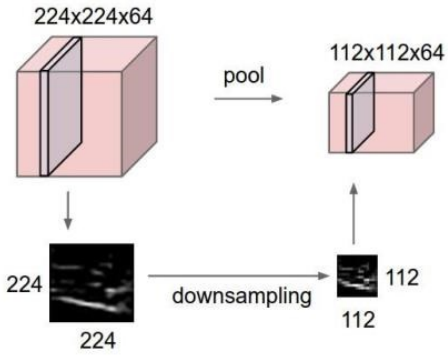


Figure 9 General pooling [22]

The reasons why we need pooling layer for image processing [21]: First, the image features can have a lower dimensionality to reduce the computational effort. The amount of information contained in an image is very large and there are many features, but some information is not useful for the image task, so the computational effort is reduced by removing such redundant information through pooling and extracting the most important features for retention. Second, there is less chance of overfitting, which is easily caused when there are too many parameters and is also more convenient to optimize. Therefore, we can use the pooling layer to decrease the size of the image and improve the computational speed. Considering the scale invariance of the features, when doing compression, only the irrelevant information is removed, leaving the features with scale invariance, thus simplifying the image's feature maps [20]. There are two common pooling methods.

Average pooling: As in Fig. 10, the mean value is taken for all pixel points in the pooled area. This method is often used to obtain background information because the obtained feature data are more sensitive to background information. We calculate the average pooling by Eq. (2).

$$y_{kij} = \frac{1}{|R_{ij}|} \sum_{(p,q)\in R_{ij}} x_{kpq}. \tag{2}$$

Maximum pooling: As seen in Fig. 10, the maximum value is taken for all pixel points in the pooling region. We calculate the maximum pooling through Eq. (3).

$$a_{kij} = \max_{(p,q)\in R_{ij}} x_{kpq} \tag{3}$$

This method is often used to obtain texture feature information due to the obtained feature data being more sensitive to texture feature information. The role of the convolution layer is to obtain the local features of the previous layer, while the role of pooling is to merge similar features with the purpose of dimensionality reduction.

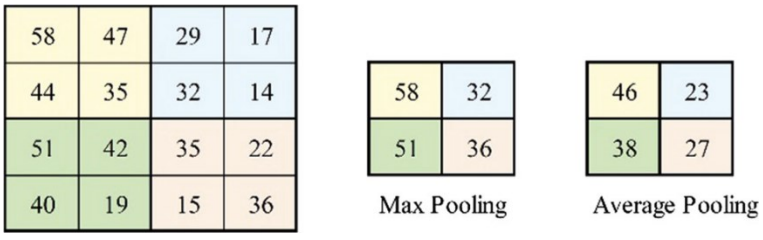


Figure 10 Output feature after applying max and average pooling [22]

Full-connection layer: It is similar to a traditional neural network. The role of the fully connected layer is to connect all neurons and pass data to the next layer of neurons, where each neuron in the previous layer is interconnected with the neurons in the next layer. It is called fully connected layer because all local features are used. The fully connected layer generally follows all the convolutional and pooling layers, before the output layer, to classify the data.

Activation Layer [24][25]**:** The activation function serves to selectively feature-enhance or feature-weaken the neuron nodes, enhancing the activation of useful target features and diminishing the useless ones, which can solve nonlinear problems.

In the absence of an activation function, the output of each layer is a linear function of the input of the layer above it, and the result is a linear combination of the sources regardless of the number of layers in the neural network. This situation is the most primitive perceptron.

When we use the activation function, which provides a nonlinear element to the neurons, this enables the neural network to approximate any nonlinear function, hence enabling the neural network to extend to ta variety of nonlinear models. In addition to this, a more robust representation can be obtained.

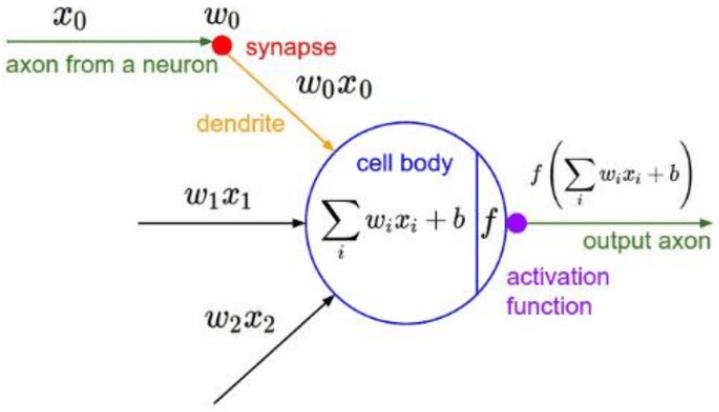


Figure 11 Mathematical model of a neuron in a neural network [24]

Activation function:

Sigmoid function: The Sigmoid function is an S-shaped function commonly found in biology, also known as an S-shaped growth curve. In information science, the Sigmoid function is often used as a threshold function for neural networks, mapping variables to between 0,1, due to its single-increasing as well as inverse single-increasing function properties. The formula and image are as follows:

$$\begin{aligned} \text{sigmoid} = f(x) &= \frac{1}{1+e^{-x}} \\ f'(x) &= f(x)\big(1 - f(x)\big) \end{aligned} \tag{4}$$

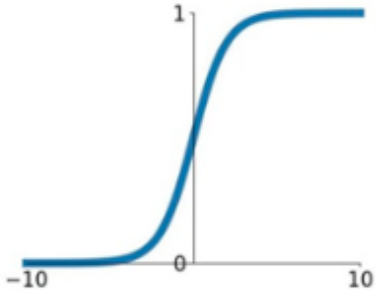


Figure 12 Sigmoid function [24]

The advantage of Sigmoid function is that the range of values (0, 1) are relatively simple and easy to understand. The disadvantage is easy saturation and termination of gradient transfer.

Tanh function: Tanh is one of the hyperbolic functions, and $\mathrm{Tanh}(\times)$is the hyperbolic tangent. In mathematics, the hyperbolic tangent "Tanh" is derived from the hyperbolic sine and hyperbolic cosine of the basic hyperbolic function. The Eq. (5) and images (Figure 13) are as follows.

$$tanh = f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$$
$$f'(x) = 1 - f(x)^{2} \tag{5}$$

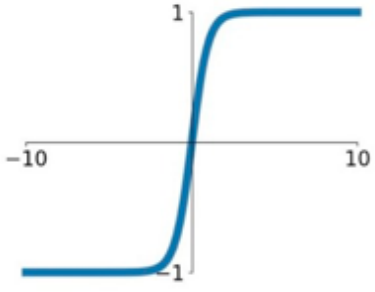


Figure 13 Tanh function [24]

The advantages are the range of values (-1, 1) easy to understand and the output of the function is centered on 0. The disadvantage is it is prone to saturation and termination of gradient transfer.

ReLu activation function: The Rectified Linear Unit (ReLu) for the output of the hidden layer neurons. Equation (6) and Figure 14 are as follows:

$$ReLU = f(x) = max(0, x) f'(x) = \begin{cases} 1, x > 0 \\ 0, x \leq 0 \end{cases} \quad (6)$$

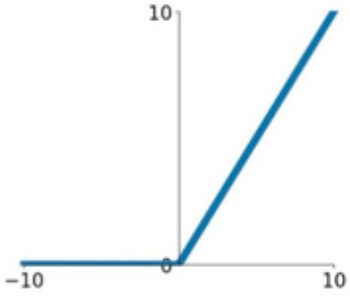


Figure 14 Relu activation function [24]

The advantages are improved convergence speed compared to the above two functions, simple gradient solver formula, and no gradient disappearance or gradient explosion.

The disadvantages are no boundary, more fragile, and easy dead-neurons which can be solved by decreasing the learning rate.

Output layer: The output layer generally follows the fully connected layer, and the output layer generates the probability of each image class through the excitation function of the fully connected layer.

Command Backbone: Convolutional Neural Networks are utilized for feature extraction. There are several CNNs available, including AlexNet, VGGNet, and ResNet. These networks are primarily employed for object classification tasks and assessed on well-recognized benchmarks and data sets, such as ImageNet. When classifying or recognizing an image, the classifier assigns a category to each image and determines the likelihood that the image belongs to that category. In contrast, for object detection, the model must be able to identify several items inside a single image and provide the corresponding coordinates. This

demonstrates that object identification can be more challenging than picture categorization [26]. Hence, the backbone's selection for our fusion task is critically important.

The advancements in image classification or object detection in images using deep learning, which were the first steps in computer vision, are primarily responsible for the exponential growth of autonomous driving technologies in this round. The perceptual algorithms can be divided into two main categories. The semantic segmentation and the object detection are two tasks that are expected to be completed. An environment perception system's objective is to produce a striking segmentation result in the 3D area surrounding the vehicle. The other type is end-to-end, where a waypoint serves as the output target. Therefore, study in the sensor fusion area is essential.

1.4. Thesis Contribution

This thesis focuses on environmental sensing systems and will target the two main sensors, camera and LiDAR and their fusion on the feature level to increase the context and the efficiency of the fusion via end-to-end training. The author designs and implements five enhanced versions of Transfuser with different fusion methods.

The main structure of this thesis is follows:

Chapter 1: This chapter includes the basic introduction to the technologies used in autonomous driving today，the importance of sensor fusion, and the basic convolutional network architecture.

Chapter 2: This chapter introduces the development of fusion technology, deep learning-based image processing methods, deep learning-based LiDAR signal processing methods, and existing autonomous driving perception technologies using LiDAR and camera fusion algorithms.

Chapter 3: This chapter covers the methods used in our designed neural network model and a detailed explanation of our designed and proposed model.

Chapter 4: This chapter includes the experimental design, experimental process, evaluation metric, experimental analysis, and the simulation results of fusion models.

Chapter 5: This chapter summarizes our design models and the future work.

CHAPTER 2 LITERATURE REVIEW

The LiDAR, cameras, and millimeter-wave radar are the primary sensors utilized in autonomous driving perception technologies. Each of these sensors has its advantages and disadvantages and complements each other, so how to efficiently fuse multi-sensor data has naturally become one of the popular areas for perception algorithm research. This chapter describes fusing LiDAR and camera in perception tasks and focuses on the current primary deep learning-based fusion algorithms.

The camera generates 2D pictures that are highly accurate for discerning the form and category of objects. The success of deep learning techniques began with computer vision applications. The processing of image data also serves as the foundation for many efficient algorithms. Therefore, the current image-based perception approaches are relatively developed. However, collecting depth (distance) information about scenery and objects is also challenging. LiDAR effectively compensates for the limitations of cameras and can accurately sense the distance of objects.

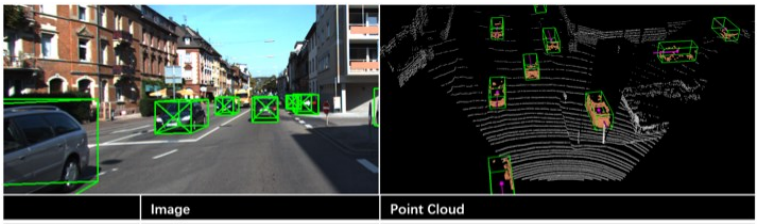


	Image	Point Cloud
Permutation	Ordered	Orderless
Data Structure	Regular	Irregular
Data Type	Discrete	Continuous
Dimension	2D	3D
Coordinates	Projective	Euclidean
Resolution	High	Low

Figure 15 a comparison between image data and point cloud data [27]

Fig. 15 shows the huge differences between image data and point clouds [27]. First, while the three-dimensional point cloud includes three-dimensional information in the coordinate system and may be projected to a range of viewpoints, the picture data is a two-dimensional representation of the real world created by projection. Second, the data structures are different; image data are regular, ordered and dense, while point cloud data are irregular, disordered and

sparse. The image data's spatial resolution is also much higher than point cloud data. Therefore, extracting the data feature from the point cloud and image data accurately is key to our fusion.

2.1. Deep-learning Base for Image

We are using deep learning to extract the features. The CNN-based methods are from the original AlexNet, and VGGnet, to a smaller volume of Inception, ResNet series, and then DenseNet series. All of them are based on deep learning methods, extracting features from CNN networks to learn a feature map for images.

AlexNet: In order to improve learning ability, traditional networks have used random sparse connections. The computer hardware and software had poor computational efficiency for non-uniformly sparse data. However, as the actual training target diversity became richer, the sample size of the labelled dataset becomes larger. Hence, it requires higher capacity models for learning. The convolutional neural networks could control the capacity of the model by adjusting the depth and width and taking full advantage of the local spatial correlation property of images. At the same time, hardware such as GPUs and highly optimized implementations of 2D convolutional operations are powerful enough for training larger CNNs. The AlexNet was proposed in 2012, was designed by Geoffrey Hinton and his student Alex Krizhevsky, who won the 2012 ImageNet competition [28]. To address the problems of the small type of capacity of traditional models and the fact that they are not easy to use in practice and prone to overfitting. The AlexNet contains 630 million connections, 60 million parameters and 650,000 neurons, with 5 convolutional layers, 3 of which are connected behind the maximum pooling layer, and finally 3 fully connected layers. As you can see in the Fig. 16, AlexNet also

uses large convolutional kernels of 11×11 and 5×5, and the specific network structure is shown in the following Fig.16.

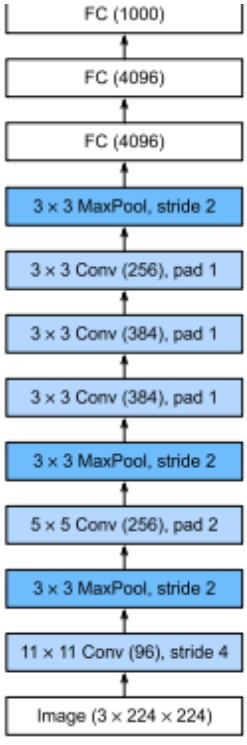


Figure 16 AlexNet network structure [27]

The main contribution points of AlexNet are:

1. Accelerated convergence by using ReLU activation functions. Previously, the main activation functions used were Tanh(x) in Eq. (5), but these are saturated activation functions (Fig. 13). The gradient is almost zero when the input is in saturation, so the convergence rate is extremely slow.

To solve this problem, the AlexNet uses ReLU in Eq. (6) as the activation function: $f(x) = max\ (0, x)$. It does not have a saturation zone; the derivative is always one, the gradient is larger, the computation is less. So, it converges faster (Fig. 14).

2. Using GPU parallelism to speed up training. It also lays the foundation for the subsequent theory of group convolution. The largest size of the network that can be trained and the batch size are both constrained by the fact that a single GTX 580 GPU only has 3GB of video memory. Therefore, it divides the model into two parts and distributes them to two GPUs for

training. Parallelism can be easily performed since data can be exchanged directly between the GPUs without going through the host's memory

3. Local Response Normalization (LRN) is proposed to increase the generalization property. In neurobiology, there is a concept called lateral inhibition, which refers to the inhibition of adjacent neurons by activated neurons. Normalization aims to "inhibit, " and local normalization draws on the idea of lateral inhibition to achieve local inhibition.

This lateral inhibition is useful when using ReLU because the response of ReLU is unbounded and therefore needs to be normalized. The use of a local normalization scheme helps to increase the generalization capability.

The core idea of LRN is to normalize using the nearest neighbor data, shown in Eq. (7).

$$b_{x,y}^{i} = a_{i,x,y}^{i} / \left(k + \alpha \sum_{j=max(0,i-n/2)}^{min(N-1,i+n/2)} \left(a_{s,y}^{j} \right)^{2} \right)^{\beta} \tag{7}$$

Where K, n, α,β is the hyperparameter, x, y denotes the value at the (x, y) position on the i^{th} convolution kernel, N is the total number of channels, and b is the normalization result. The LRN allows the points with large local response values to suppress the points with small local response to enhance feature comparison.

4. Using overlapping pooling to prevent overfitting. In general pooling, the pooling window equals the sliding step. And overlapping pooling refers to the pooling (Equation 3) when there is an overlap between adjacent sliding windows. The AlexNet uses overlapping pooling, which is less prone to overfitting.

5. Proposing dropout and data augmentation to prevent overfitting. Combining the predicted values of different models is a good way to reduce the testing error, but it is extremely expensive. Therefore, using the dropout technique, the value of each hidden neuron is set to 0 with a probability of 0.5. The dropout neurons will not participate in forwarding and backward propagation.

The dropout is repeated for each forward propagation in the training phase. Thus, the model

is randomly sampled into different architectures each time there is a new input, but all the architectures share the weights. This technique reduces the interdependence between neurons. Thus, the model is forced to learn more robust features.

Visual Geometry Group (VGG) [29]**:** The VGG is a good feature extractor. Compared with the Alexnet, VGG has the following improved points:

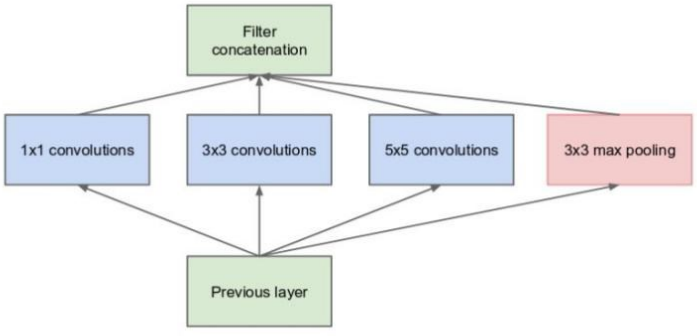
1. Removing the LRN layer, find that the role of LRN in deep networks is not obvious and simply eliminated.

2. The VGG has smaller convolutional kernels 3×3, while the AlexNet uses larger convolutional kernels, so the VGG has fewer parameters compared to the AlexNet.

3. The pooling kernel becomes smaller, the pooling kernel in the VGG is 2×2 and the stride is 2; for the AlexNet the pooling kernel is 3×3 and the stride is 2.

In order to better explore the effect of depth on the network, the problem of the number of parameters must be solved. A deeper network means more parameters and more difficulty to train, especially when using large convolutional kernels. Through their analysis, the authors concluded that due to the nature of convolutional neural networks, a convolutional kernel of size 3x3 is sufficient to capture the variation of horizontal and vertical as well as diagonal pixels. Using large convolutional kernels would bring an explosion in the number of parameters not to mention that there would be some parts of the image that are convolved multiple times, which may cause difficulties in feature extraction. So, in VGG a 3×3 convolution is commonly used. In addition, three fully connected layers are used in the last few layers of the VGG network, which are eventually connected to a softmax. In fact, the parameters of these three fully connected layers occupy a large part of the overall parameters of VGG, but at present, in order to reduce the number of parameters, the fully connected networks of the latter layers are pooled by global average pooling convolution. It uses convolutional operations instead, but global average pooling also has a great advantage. That is, the higher up in the neural network, the sparser the network should be, while being more expressive. This avoids the problem that too-deep neural networks are too computationally intensive and prone to overfitting.

Inception V1-V3 [30][31][32]**:** First, choosing the appropriate kernel size for the convolution procedure becomes challenging because of the enormous variance in information placement. Smaller kernels are useful for acquiring local information, but larger kernels are useful for gathering global information. Second, overfitting may happen in extremely deep networks. The whole network must be updated with gradient information, which is equally challenging. On the other hand, the computation cost will be very expensive when the network is simply stacked. Therefore, instead of becoming "deeper" the network becomes "wider." Fig. 17 below shows the "simple" Inception module. It performs a convolution input with 3 different filter sizes $(1 \times 1, 3 \times 3, 5 \times 5)$. In addition, it also performs maximum pooling. They combine their outputs and send it to the next Inception module.



(a) Inception module, naïve version

Figure 17 Inception module [28]

As mentioned before, the cost of computing deep neural networks is very expensive. Hence, to reduce the cost, the authors suggest reducing the number of input channels and including a 1×1 convolution before the 3×3 and 5×5 convolution blocks. Although adding an extra operation seems to result in more computation, the cost of 1×1 convolution is in fact much cheaper than the 5×5 convolution, and the reduced number of input channels results in less computation.

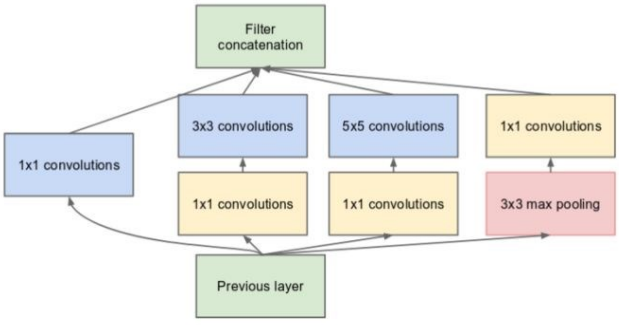


Figure 18 Inception module with dimension reductions [28]

The Inception module builds the neural network architectures using reduced dimensions. This is often referred to as the GoogLeNet (the Inception v1). The GoogLeNet has a linear stack of 9 such the Inception modules. The neural networks perform better when the convolution does not significantly reduce the dimensionality of the input. The excessive dimensionality reduction may lead to information loss, which is called a "representational bottleneck."

To reduce the representational bottleneck, [29][30] proposed four principles for designing the network as shown below:

1. Do not compress and reduce the dimensionality too early to avoid the loss of information representation (avoiding the bottleneck of information representation).

2. Enhance high-dimensional representation (3×3 split into 1×3 and 3×1 convolutional kernels juxtaposed in the high-level Inception structure becomes wider to enhance information representation).

3. Dimensionality reduction can be achieved with 1×1 convolutional kernels before convolution (the authors argue that information within the same channel is highly correlated, so dimensionality reduction at the channel level does not have much detrimental effect and also speeds up training).

4. Pay attention to the balance of network width and depth.

The Inception-v2 is similar to the VGG in that the large convolutional kernel is split into several small convolutional kernels. At the same time, asymmetric convolution is introduced.

The reason not to reduce the 3×3 convolution kernel to two 2×2 convolution kernels is there are fewer convolution parameters using this asymmetry of 1×3 superimposed on 3×1. The Inception-v2 also adopts a more efficient data compression in order to compress the size of the feature map to half, while the number of channels becomes twice as many. The authors use a reduction structure similar to the Inception module, doing both pooling and convolution. The stride is 2, and then stack the two results to achieve the compression of the feature map and the expansion of the channels.

2.2. Deep-learning Base for Cloud Point

The output data of LiDAR is a 3D point cloud, where each point contains not only X, Y, and Z coordinates, but also a reflection intensity R. Since LiDAR data is a relatively sparse point cloud as opposed to the dense grid structure of pictures, it is necessary to adapt techniques typically employed in the image domain to work with point cloud data. To take advantage of algorithms in the image domain, the point cloud can be converted to a dense grid structure in bird's eye view or Range View. In addition, the Convolutional Neural Network (CNN) in deep learning can also be improved for sparse point cloud structures, such as the PointNet [33]. Depending on the organization of the input data, the processing methods can also be divided into point-based methods, grid-based methods, and projection-based methods.

VeloFCN [39] is one of the representative methods for processing point cloud data, which naturally draws on successful experiences in the field of vision. It converts a 3D point cloud to a front view similar to an image and obtains a "point cloud pseudo-image." This data is very similar to the image in terms of format and properties, so it is natural to copy the algorithm from the image. However, the drawbacks of this representation are obvious. Multiple points may be mapped to the same location of the image coordinates, which may cause information loss. As well, mapping the 3D points to the 2D plane, the depth information is embedded in the pixel values and the extraction of 3D information becomes relatively difficult. Therefore, the author thought to map the 3D point cloud to the bird's eye view. This mapping is very intuitive and can be simply thought of as ignoring the height coordinates of the 3D points (treating them as features of the points) to obtain a representation of the data in the 2D plane. MV3D [36] is the mapping of the 3D point cloud to both the front and top views and fusing it with the 2D image data. All the above mentioned are data construction and feature extraction.

VoxelNet [41] and PointNet++ [42] represent two basic directions of point cloud processing, VoxelNet quantifies point clouds as mesh data, while PointNet++ directly deals with unstructured data points. The idea of VoxelNet [41] is not complicated, first quantizing the point cloud into a uniform 3D grid. Each point is represented by a 7-dimensional feature,

including the X, Y, and Z coordinates of the point, the reflection intensity R, and the position differences ΔX, ΔY, and ΔZ of the point relative to the center of mass of the grid (the mean of all point positions in the grid). A fully connected layer is used to extract the point features, and each point feature is then stitched with the mean of all point features in the grid to obtain a new point feature. The new point features are then obtained by stitching each point feature with the mean of all the points in the grid. The advantage of this feature is that it preserves both the characteristics of a single point and the characteristics of a small local area (the grid) around that point. This point feature extraction process can be repeated several times to enhance the descriptive power of the feature. Finally, all points within the grid are subjected to a Max Pooling operation to obtain a fixed length feature vector. These steps are called feature learning network and the output is a 4D Tensor (corresponding to X, Y, Z coordinates and features). The 3D convolution is used in VoxelNet to compress the Z dimension (e.g., stride = 2). Suppose the dimension of 4D Tensor is $H \times W \times D \times C$, after several 3D convolutions, the size of Z dimension is compressed to 2 (i.e., $H \times W \times 2 \times C'$), and then the Z dimension is directly combined with the feature dimension to generate a 3D Tensor ($H \times W \times 2C'$).

2.3. LiDAR and Camera Fusion Method for Object Detection

With the popularization of LiDAR and the opening of LiDAR database, the development of point cloud data processing research. The following object detection task is mainly used to introduce the choice of different fusion methods of backbone. The idea of fusion methods for other tasks can be obtained from it. The main idea of this strategy is to generate a candidate frame (proposal) of the object by one kind of data first.

A 2D candidate frame is produced if picture data is utilized, and a 3D candidate frame is produced if point cloud data is used. Then, the candidate frame is combined with another kind of data to generate the final object detection result (the data used to generate the candidate frame can also be reused). This combination process is to unify the candidate frame and data under the same coordinate system, either 3D point cloud coordinates (e.g., F-PointNet) or 2D image coordinates (e.g., IPOD).

From picture data, F-PointNet [33] creates 2D object candidate frames, which are subsequently projected into 3D space. All points falling inside the optic vertebra are integrated as the characteristics of each 2D candidate frame, which corresponds to a 3D optic vertebra (frustum). To eliminate these distractions and retain just the points on the objects for further object frame estimation, 3D instance segmentation is required. The points in the visual vertebrae may originate from occluded objects in the front or background (similar to the processing in PointNet). The disadvantage of this view spine-based approach is that only one object to be detected can be processed in each view spine, which is not sufficient for crowded scenes and small targets (e.g., pedestrians).

To address the above problems of the visual vertebrae, IPOD [34] proposes the use of 2D semantic segmentation to replace 2D object detection. By projecting the point cloud into the 2D image space, the background points from the point cloud are first removed using the picture's semantic segmentation findings. At each foreground point of interest, candidate object frames are then created. Next, overlapping candidate frames are removed using NMS, and then around 500 candidate frames are maintained per point cloud frame. The extraction of point features is done while using the PointNet++ grid. The last stage uses a small-scale PointNet++ to forecast the category and the precise object frames using the candidate frames and point characteristics. IPOD generates dense candidate object frames based on semantic segmentation, so it works better in scenes containing a large number of objects and mutual occlusions.The previous two techniques produce candidate frames using outcomes from object detection and semantic segmentation on 2D pictures, respectively, and then only handle point cloud data after that.

With the rapid development of 3D object detection technology, the selection of object candidate frames has gradually changed from 2D to 3D. MV3D [36] is a representative work based on 3D candidate frames. It starts by mapping the 3D point cloud to the Bird’s Eye View (BEV) view and then creates candidate frames for 3D objects based on this view. These 3D candidate frames are then mapped to both the picture view and the front view of the point cloud, and the appropriate features are fused. By pooling Region of Interest Pooling (ROI) based on the candidate frames, the feature fusion is accomplished.

AVOD [37] is also intended to combine picture and point cloud data using 3D candidate frames. However, the original candidate frames are not generated by point cloud processing, but by a priori knowledge of uniform sampling in BEV view (0.5 m interval, size is the mean value of each object class). The point cloud data is used to assist in removing empty candidate frames, so that eventually 80,000 to 100,000 candidate frames are generated per frame of data. These candidate frames are further filtered by the fused image and point cloud features and then sent to the second stage of the detector as final candidates.

Feature Layer Fusion: The late fusion is characterized by fusing different features with object candidate frames as the center, and the fusion process usually uses ROI pooling (e.g., bilinear interpolation). This operation leads to the loss of spatially detailed features. Another idea is feature layer fusion, which means directly fusing multiple features.

The point cloud is mapped to the image space and merged with the RGB channel of the image as an additional channel with depth information. This idea is simple and straightforward and works well for two-dimension object detection. But the fusion process loses a lot of 3D spatial information, so it does not work well for 3D object detection. Due to the rapid development in the field of 3D object detection, feature layer fusion works best with 3D coordinates, which can provide more information for 3D object detection.

The continuous convolution (CC) is used by ContFuse [38] to combine the characteristics of point clouds and images, and the fusion operation is under the BEV perspective. For a pixel (grid) on the BEV, the image features of each point are obtained by the K nearest neighboring points in the point cloud data and then mapping these points in 3D space to the image space. The XY offset of each point in relation to the matching BEV pixel is its geometric characteristic. In order to acquire the feature values at the relevant BEV pixels, the geometric and image characteristics are integrated as point features, which are then weighted and summed in accordance with the sequential convolution technique (the weights rely on the XY offsets). Similar processing is applied to each BEV pixel to produce a BEV feature map. As soon as picture features are converted to BEV views, they may be simply combined with BEV

features from point clouds. To enhance the detection of objects of various sizes, ContFuse performs the aforementioned feature fusion at various spatial resolutions.

2.4. Multimodal Fusion

The multimodal fusion is a very critical research point in the multimodal study, which integrates the information extracted from different modalities into a stable multimodal representation. The multimodal fusion and representation are clearly related, and if a process is focused on integrating different unimodal representations using some architecture, then it is classified as fusion. Because early and late fusions suppress intra- or inter-modal interactions, research now focuses on intermediate fusion methods, allowing these fusion operations to be placed in multiple layers of a deep learning model. There are three main approaches for fusing text and images: simple operation-based, attention-based, and tensor-based approaches [51].

Simple operational fusion approach: The feature vectors from different modalities can be integrated by simple operations, such as splicing and weighted summation. Simple operations make little connection between parameters, but subsequent network layers automatically adapt themselves to such operations. There are two ways to combine feature maps, one is to add the elements correspondingly and the other is to stack feature maps together.

Concatenation: The concatenation operation can be used to combine the input features [43][44][45] of the lower layers or the features [45][46][47] of the higher layers (features extracted by the pre-trained model) with each other.

Weighted sum: For weighted summation methods where the weights are scalars, this iterative approach requires that the vectors generated by the pretrained model have a defined dimensionality and be in a certain order and suitable for element-wise addition [48]. To meet this requirement fully connected layers can be used to control the dimensionality and to reorder each dimension.

Attention based approach to fusion: A lot of attention mechanisms have been applied to fusion operations. An attention mechanism usually refers to a weighted sum of a set of scalar weight vectors dynamically generated by a set of "attention" models at each time step [47][48]. The multiple output heads of this attention set can dynamically generate the weights to be used in the summation, so that additional weight information can be preserved in the final stitching. When applying the attention mechanism to an image, different weights are applied to the image feature vectors of different regions to obtain a final overall image vector.

For the attention algorithm [40], the required inputs are Q, K, V. And depending on the source of Q, K, V, it can be further called self-attention or cross-attention. The self-attention means that Q, K, V originate from the same place, e.g., X, and there are in Eq. (8).

$$\begin{aligned} Q &= W^Q X \\ K &= W^K X \\ V &= W^V X \end{aligned} \tag{8}$$

The cross-attention [50] means that Q, K, and V originate from different places. We can define the Q, K, V as Eq. (9).

$$\begin{aligned} Q &= W^Q X_1 \\ K &= W^K X_2 \\ V &= W^V X_2 \end{aligned} \tag{9}$$

Taking scaled dot product attention as an example, its mathematical form is Eq. (10).

$$(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \boldsymbol{softmax}\left(\frac{\boldsymbol{QK^T}}{\sqrt{\boldsymbol{d_k}}}\right)\boldsymbol{V} \tag{10}$$

2.5. Research Gap

In the field of autonomous driving, there are many streams of development. For example, Tesla uses cameras as the main sensor autonomous driving system, and another mainstream

direction is Google's Waymo, whose research direction mainly uses lidar and camera fusion. Each stream makes important and unique contributions to the literature on how to effectively fuse information from different sensors to enhance the acquisition of valuable information during autonomous driving. Thus, we need to increase the efficiency and effectiveness of sensor fusion. Our research focuses on environmental sensing systems fusion methods like LiDAR and camera fusion. We have major studies on the Transfuser, which is a new end-to-end multimodal fusion approach proposed in 2021. The Transfuser method is an innovative multi-modal fusion transformer that combines image and LiDAR representations by paying attention.

The Transfuser [56] fusion model targets at the two main sensors, camera and LiDAR, based on the CNN architecture. After lidar and image feature extraction using the ResNet backbone network, a self-attention fusion mechanism is used to extract the environmental information. The Transfuser shows that imitation learning rules based on current sensor fusion techniques cannot handle adversarial situations in urban driving, such as unprotected turning at crossings or people appearing from obscured areas. The Transfuser suggests a unique multi modal fusion transformer to add the global context of the 3D scene to the feature extraction layers of the different modalities.

However, we find that the Transfuser [56] method simply concatenates the feature maps together when fusing, which is not efficient and effective for feature fusion. There is some information lost, like the details of context and some features. As a result, we propose five new fusion architectures and methods to improve the efficiency and effectiveness of fusion.

CHAPTER 3 METHODOLOGY

In this chapter, we present the deep-learning network for camera and LiDAR. The method contains the multimodal fusion model (Backbone Network) part and auto-regressive waypoint prediction task (Fig. 19). We will focus on mid-fusion after the feature extraction and introduce the components of the backbone network in this section. In order to compare the efficiency of the fusion model, we are using the same waypoints decoder as the Transfuser [56].

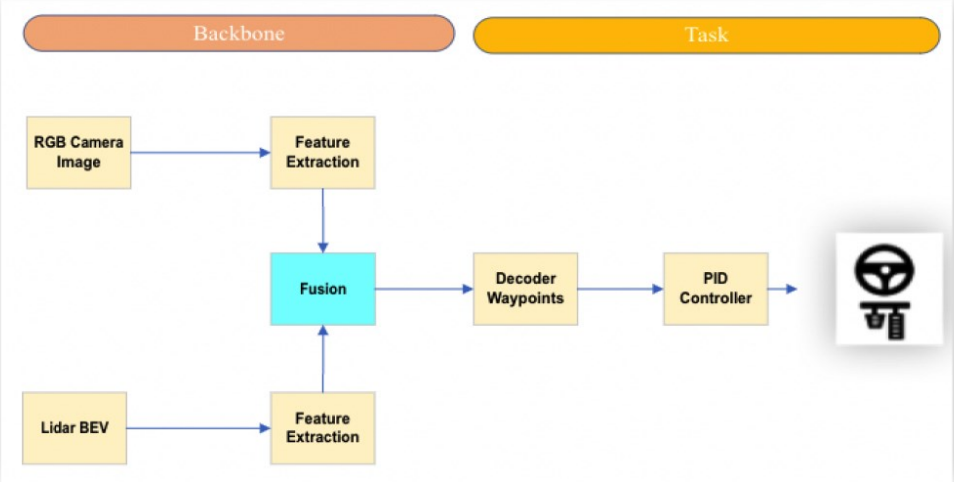


Figure 19 End-to-end driving architecture

3.1. Sensor Fusion Backbone Network

In neural networks, especially in the field of computer vision, we use the appropriate main core for performing image feature extraction, which is the basis of the whole fusion and model. We use the extracted feature maps as input for downstream tasks, which include classification, object detection, and waypoint prediction tasks. [57]. In our paper, the task of the self-driving car is to generate a predicted movement path.

In our model, we selected the ResNet to do the feature extractor (Figure 21). Why Resnet? In the CNN Architecture, the different layers automatically learn to extract some features, such as, low-level visual features. Hence the engineers aim to extract more feature information;

they might add more CNN layers in their design model. However, when network depth increases, accuracy drops. In other words, the deeper model is degradation. Fig. 20 shows that the 56-layer network is less effective than the 20-layer network. This would not be an overfitting problem, because the training error of the 56-layer network is equally high. We know that deep networks have gradient disappearance or explosion problems, which make it difficult to train deep learning models.

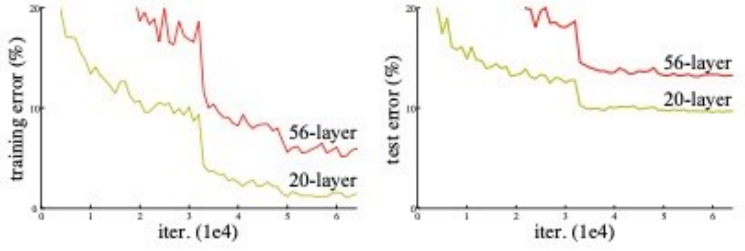


Figure 20 Training error caused by deeper layer [54]

The idea of deep residual learning originated from the paper [54]. This paper proposes that if there exists the current optimal layer of the network N, then a deeper network whose last few layers are identity mapping of the output of the optimal layer of that network N can be constructed to achieve results consistent with the optimal layer; if it is not yet the optimal layers, then a deeper network can achieve better results.

Dr. He raised using a deep residual learning model to deal with this situation [54]. The structure of the residual learning block is shown for a stacked-layer structure as Fig. 21, when the input is x, the learned features are marked as $\mathcal{H}(\mathrm{x})$. Now we want it to learn the residual $\mathcal{F}(\mathrm{x}) = \mathcal{H}(\mathrm{x}) - \mathrm{x}$, so that the original learned features are actually $\mathcal{F}(\mathrm{x}) + \mathrm{x}$. When the residual is zero, the stacking layer uses an identity mapping [55] to make sure the network performance will not degrade at this point. In fact, if the residual is not zero, it will allow the stacked layers to keep learning new features based on the input features for better performance.

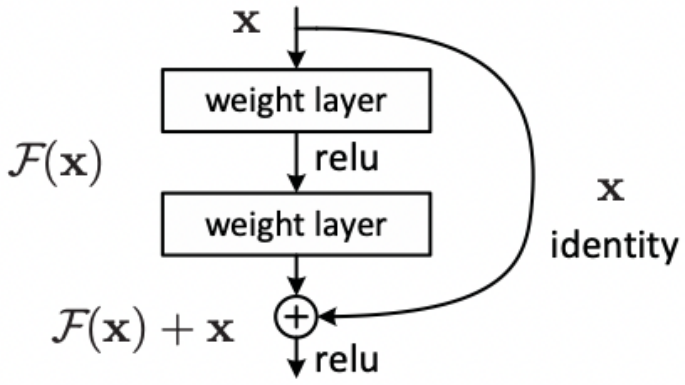


Figure 21 Residual learning block [54]

Table 2 Parameter table for ResNet architecture [54]

Layer name	Output size	18-layer	34-layer	50-layer	101-layer	152-layer
Conv1	112×112	7x7, 64. stride 2				
conv2_x	56×56	3x3 max pool, stride 2				
		$\begin{bmatrix}3\times3,64\\3\times3,64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3,64\\3\times3,64\end{bmatrix}\times3$	$\begin{bmatrix}1\times1,64\\3\times3,64\\1\times1,256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1,64\\3\times3,64\\1\times1,256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1,64\\3\times3,64\\1\times1,256\end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix}3\times3,128\\3\times3,128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3,128\\3\times3,128\end{bmatrix}\times4$	$\begin{bmatrix}1\times1,128\\3\times3,128\\1\times1,512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1,128\\3\times3,128\\1\times1,512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1,128\\3\times3,128\\1\times1,512\end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix}3\times3,256\\3\times3,256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3,256\\3\times3,256\end{bmatrix}\times6$	$\begin{bmatrix}1\times1,256\\3\times3,256\\1\times1,1024\end{bmatrix}\times6$	$\begin{bmatrix}1\times1,256\\3\times3,256\\1\times1,1024\end{bmatrix}\times23$	$\begin{bmatrix}1\times1,256\\3\times3,256\\1\times1,1024\end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix}3\times3,512\\3\times3,512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3,512\\3\times3,512\end{bmatrix}\times3$	$\begin{bmatrix}1\times1,512\\3\times3,512\\1\times1,2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1,512\\3\times3,512\\1\times1,2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1,512\\3\times3,512\\1\times1,2048\end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

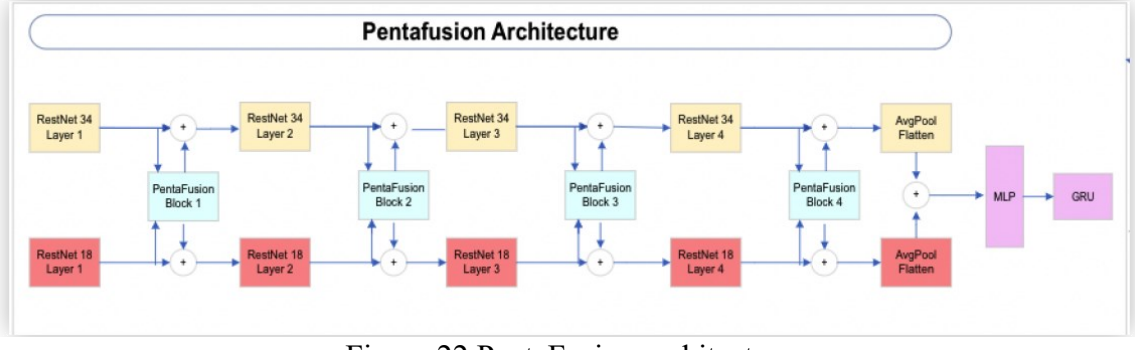


Figure 22 PentaFusion architecture

3.2. Input Representations

Figure 23 RGB image from the camera with 256 × 256 pixels

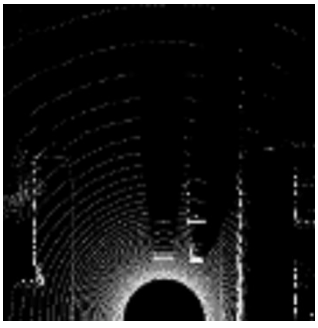


Figure 24 BEV image from LiDAR with 256 × 256 pixels

Input from the camera:

The input from the camera is RGB images obtained from the dataset front view camera, which we need to crop to 256×256 pixels before sending to do the feature extract (Fig. 23). The RGB picture input is from the front camera with a 100 Front of View (FOV). The camera is 2.3 meters above the ego-ground vehicle's level and 1.3 meters from its center. We offset the cameras by 1.3 meters to prevent the vehicle hood from obscuring rendered pictures. We get 400×300 pixel photos with this camera [58].

Input from LiDAR:

The LiDAR point cloud must be converted into a 2-bin histogram across a 2D BEV grid with a set resolution. From the LiDAR point cloud included in this 3D volume, we randomly choose five points and project them onto the picture space. Moreover, the resolution is 256×256 pixels in Fig. 24. The LiDAR sensor is installed 2.5 meters above ground level. The LiDAR is a 64 ray-cast Velodyne, which has an 85m range and 10 FPS rotation frequency. The LiDAR's top FOV and lower FOV are both adjusted to ten and thirty degrees respectively [58].

3.3. CNN Architecture Design

ResNet: In our proposed model, there are two steams in Fig. 22. We are using the four layers of ResNet 34 block to extract the RGB feature map, the intermediate output feature maps for each layer are dimensions $(H \times W \times C)$ $64 \times 64 \times 64, 32 \times 32 \times 128, 16 \times 16 \times 256$, and $8 \times 8 \times 512$ respectively. We are using four layers of ResNet 18 for LiDAR BEV to get the feature map; the size of each block is the same as the RGB branch. Due to the computation limitation, we use average pooling (Fig. 9) after the ResNet blocks to down-sample the feature map into $8 \times 8 \times 64$, $8 \times 8 \times 128, 8 \times 8 \times 256$, and $8 \times 8 \times 512$ before the fusion block.

PentaFusion model architecture：

The fusion network as shown in Fig. 25 proposed two parts: one is the five branches fusion (Inspired by BiFusion [59]) and the other is the GPT block, which is the Self-Attention mechanism in Fig. 22. For example, we obtain the $8 \times 8 \times 64$ $(H \times W \times C)$ feature map from both RGB and LiDAR through the Residual block combining the two feature maps in which the dimension is still $8 \times 8 \times 64$ $(H \times W \times C)$. After fusion, the feature map $8 \times 8 \times 64$ $(H \times W \times C)$ goes through the GPT block. We need to do an up-sample to recover to the $64 \times 64 \times 64$ original size, so we can do the element-wise addition with the previous original feature map (before fusion). The vehicle's current velocity will be projected into a C dimension by using a linear layer.

Next, the new sum-up feature map containing original and fused information will get into the next layer of Resnet 34 and Resnet 18 to do the feature extractor again. There are four layers of ResNet in our proposal in Fig. 22.

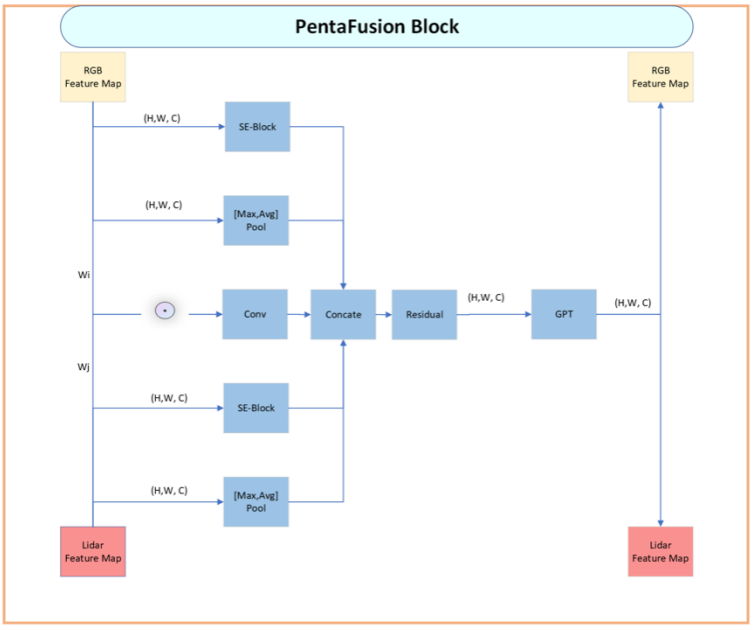


Figure 25 PentaFusion block architecture

Channel attention module SE-Net block [60]**:** The SE-Net is short for Squeeze-and-Excitation Networks. The purpose of the SE-Net block addition is to try feature extraction from the channel dimension's feature layer, which is simple to build and can be quickly loaded into the current network model framework. In order to improve the impact, The SE-Net primarily learns the connection between channels and filters out the attention for the channels with a slightly increased computation.

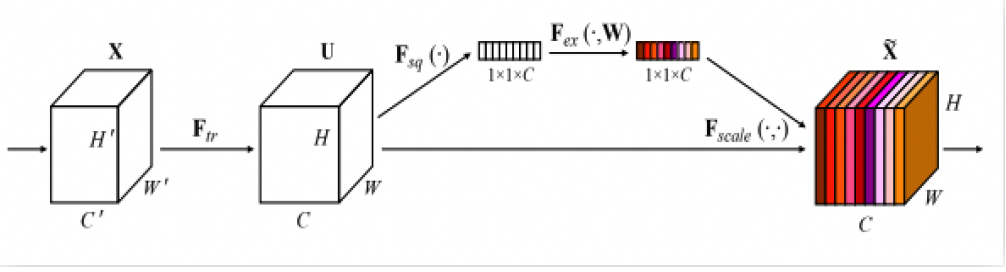


Figure 26 SE-Net block architecture [60]

Specifically, the following steps are performed: The width of the feature layer is compressed by adaptive global averaging pooling, leaving only the information of the channel dimension. Then, the channel information is the self-attention using two consecutive fully connected layers (first squeeze and then excitation to the original number of channels C). Finally, the

fully connected output is passed through a sigmoid function shown as Eq. (4) to obtain the channel weights from zero to one. Finally, the original feature layer is weighted. A global perception field may be inferred from this feature map. The paper [60] applies the SE module to the ResNet network by adding the SE-Net module to the backbone branch of the residual module as in Fig. 27.

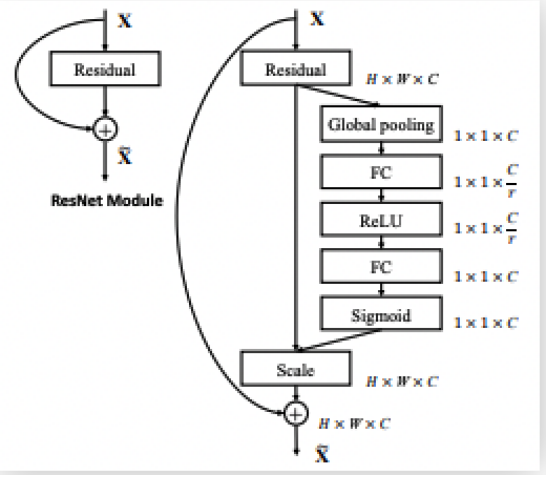


Figure 27 The structure for ResNet with SE-Net block [56]

Squeeze: The global average pooling is performed to obtain a feature map of size $1 \times 1 \times C$ from $C \times H \times W$, which can be interpreted as having a global perception field. Excitation: After Squeezing, do a nonlinear transformation on the outcome by using a fully connected neural network. The output of excitation is then multiplied by the input factors as weights.

Spatial Attention Module [61]**:** The averaging and maximizing from the channel dimension respectively, merging to obtain a convolutional layer with channel number two, and then by a convolution, and obtaining a spatial attention with channel number one. Utilizing the interspatial connectivity of features, we build a map of spatial attention. The spatial attention, which is different from channel attention, focuses on where, as an informational component, that complements the channel attention. To calculate spatial attention, we first perform average-pooling and max-pooling operations along the channel axis and concatenate the resulting feature descriptors. It has been shown that applying pooling techniques along the channel axis highlights informative areas well [62].

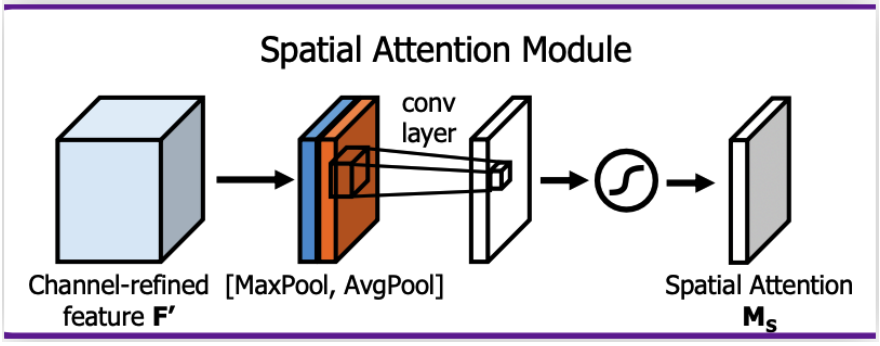


Figure 28 Spatial attention module [61]

PentaFusion block: Thus, we present the final design of the PentaFusion model as Fig. 25. We obtain the fused feature output from the PentaFusion block representation f^i by using Eq. (11), $i = 1, 2, 3, 4$.

$$
\begin{aligned}
t^i &= \text{ChannelAttn}\left(t^i\right) \\
\overline{t}^i &= \text{SpatialAttn}\left(t^i\right) \\
g^i &= \text{ChannelAttn}\left(g^i\right) \\
\overline{g}^i &= \text{SpatialAttn}\left(g^i\right) \\
b^i &= Con\,v\left(t^i W_1^i \odot g^i W_2^i\right) \\
f^i &= Residua\,l\left(\left[b^i, t^i, \overline{t^i}, g^i, \overline{g^i}\right]\right)
\end{aligned}
\tag{11}
$$

The channel attention is implemented as SE-Net block proposed to promote the global information from both LiDAR and RGB branches. The goal of the spatial attention block is to emphasize local elements and exclude unrelated areas [22]. The dot product for the b^i means the cross relationship between LiDAR and camera features maps. At this point, we obtained the feature map f^i which fused the features from both LiDAR and the image at the same time stamp. For the next stage, we need to fuse the information from different time, which considers one more dimension (sequence) into the feature map.

Transformer (the GPT block): In order to fuse the information on the time sequence, we add one dimension by using the GPT block, which uses a powerful Transformer that captures longer memory information [66].

Due to image and LiDAR modalities being complimentary, our main concept is to use the self-attention mechanism of transformers [51] to absorb the global environment. An input series of discrete tokens, each represented by a feature vector, is sent to the transformer architecture. To account for positional inductive biases, a positional encoding is added to the feature vector.

The input sequence is formally referred to as $\mathbf{F}^{in}$ where N is the sequence's token count, and each token is represented by a feature vector with dimensions D. The transformer computes a collection of queries, keys, and values using linear projections (Q, K and V) are weight matrices. Each query's results are aggregated after the attention weights are calculated using the scaled dot products between Q and K, which are obtained from $Q = \mathbf{F}^{in}M^{q}$; $K = \mathbf{F}^{in}M^{k}$; $V = \mathbf{F}^{in}M^{v}$.

$$\boldsymbol{A} = softma\,x\left(\frac{\boldsymbol{Q}\boldsymbol{K}^{T}}{\sqrt{D_k}}\right)\boldsymbol{V} \tag{12}$$

The transformer calculates the output features, $\mathbf{F}^{\text{out}}$, which have the same shape as the input features, using a nonlinear transformation $\mathbf{F}^{in}$, and the $\boldsymbol{A}$ by using Eq. (12).

$$\boldsymbol{F}^{\text{out}} = ML\,P(\boldsymbol{A}) + \boldsymbol{F}^{in} \tag{13}$$

3.4. Waypoint Prediction Network

As we can see, the task for the part is to predict the vehicle driving path point location. We simulate the test in a BEV. Hence, we assume the output position in Eq. (13) is a two-dimensional location value. The decoder that we used is shown in Fig. 29. We take the information from the GTP block, which is a 512-dimensional feature map vector that contains LiDAR and image fused sequence information as input. In order to reduce the amount of calculation, we pass the input into an MLP block and compress the vector from 512-D into 64-D. Next, we input the 64-D fused vector into the Gated Recurrent Units (GRU) block [63] and output 64-D information, which can be converted to 2-D position information, which means a prediction of the differential location point at time step $t1$. We use the current

location $\mathbf{w}_t = (0,0)$ and the predicted difference $\mathbf{\Delta w}_{t1}$ to calculate the waypoint $\mathbf{w}_{t1}$. The $\mathbf{w}_{t1}$will be the input passed into the next GRU block to get the expected waypoint, $\mathbf{w}_{t2}$. In our case, we have the T = 4 time-step, and $\mathbf{w}_t$ is the summary for the location. We used T = 4 to predict the location of the waypoint in the next four time-steps, $\mathbf{w}_{t4}$. We send the waypoint information into the PID controller to look for the value of the vehicle's brake, steering and throttle to control the vehicle behavior [56].

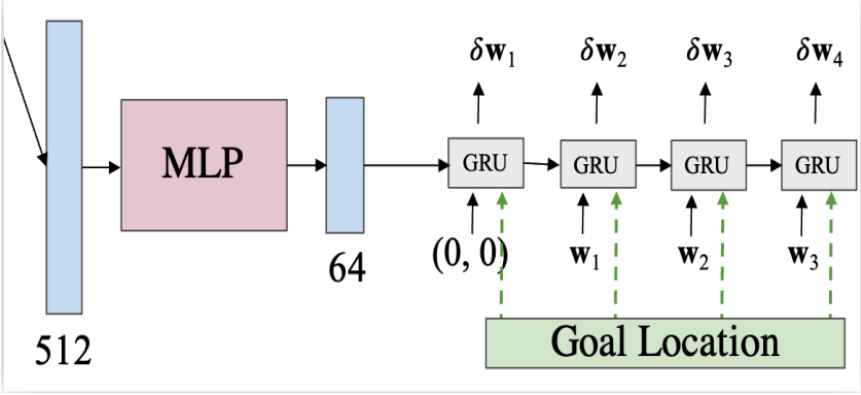


Figure 29 Waypoint Prediction Decoder module [56]

3.5. Loss Function

We use a $\mathcal{L}$ loss to train the network by comparing the predicted waypoints to the provided ground truth waypoints in the current coordinate frame. The loss function is calculated as in Eq. (14).

$$\mathcal{L} = \sum_{t=1}^{T} \| \boldsymbol{w}_t - \boldsymbol{w}_t^{gt} \|_1 \tag{14}$$

3.6. Related Work

In addition to PentaFusion model, we also designed other models. For instance, we used Convnet [10] instead of ResNet as the backbone network for feature extraction. We designed cross-attention instead of self-attention module for fusion. The simulation driving performance of these models did not perform well. The design code and experimental data

can be found on the GitHub. The rest design models can be found in Appendix A.

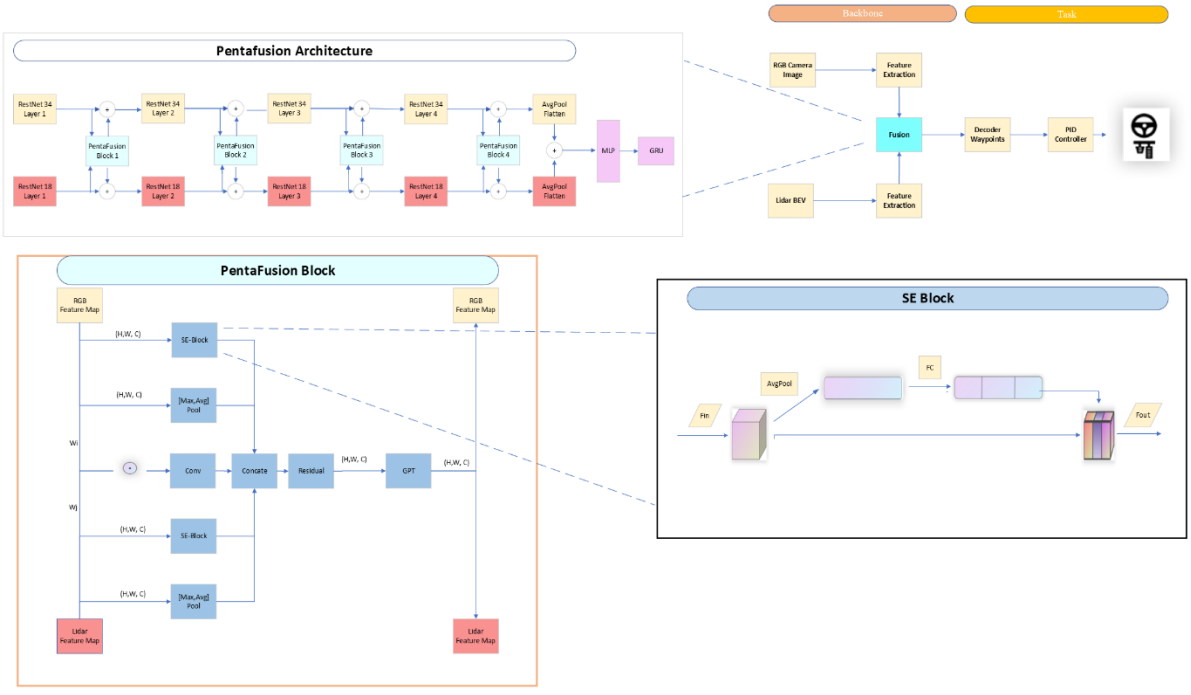


Figure 30 The overview of sensor fusion in self-driving vehicle

CHAPTER 4 EXPERIMENTATION AND DISCUSSION

In this chapter, we demonstrate the process of the experimentation setting, how to train the designed model and conduct closed-loop validation experiments on the CARLA Simulator, which includes the introduction of the dataset, the experimental environment, the experimental design, and the experimental results. We also do comparison experiments to compare the results of five other model experiments such as channel enhancement and spatial enhancement module to change the insertion position.

Dataset

We use the open-source dataset from CARLA Leatherboard as training and evaluation data, which contains 50 routes for training and 26 routes for evaluation. It was originally provided for the 2019 CARLA Challenge competition [58].

In order to have a better comparison of fusion-model performance between our design and the baseline, we use Transfuser [56] as the baseline, which was one of the winners of the 2021 CARLA Challenge. We chose the same towns, weather, and scenario setting as the baseline to do the training and evaluation. The detail of the dataset configuration can be checked from [58]. The seven towns and 14 weather variables (Table 3) are used to generate training data. We use the collection of routes and situations established and save data at a rate of two FPS. Every 30 seconds, we systematically alter the weather condition for each path.

Table 3 Provided weather conditions on the CARLA dataset map

ClearNoon	ClearSunset	CloudyNoon	CloudySunset	WetNoon
WetSunset	MidRainyNoon	MidRainSunset	WetCloudyNoon	WetCloudySunset
HardRainNoon	HardRainSunset	SoftRainNoon	SoftRainSunset	

CARLA Leatherboard provides two versions of the dataset we can use [58]:

- The minimal dataset is the 63G data, which includes only the RGB front view signal information, LiDAR signal information and measurements from the 14 kinds of weather.
- The large dataset has the 406G, which comprises multi-view camera data with various affordances and perceptual labels for both the clear weather data and 14 weathers data to aid in the construction of imitation learning agents.

In our case, we chose the first minimal size dataset version. The 63G data contains both LiDAR and front camera information, which is all we need for our sensor fusion. The data's structure and explanation are shown in Table 4.

Table 4 The data structure explanation [58]

Variable Name	**Explanation**
TownX_{tiny,short,long}	according to several localities and route files
routes_X	includes information on a certain route
rgb_{front, left, right, rear}	photos from a multi-view camera with a 400x300 resolution
seg_{front, left, right, rear}	matching segmentation pictures
depth_{front, left, right, rear}	matching depth pictures
lidar	3d point cloud in .npy format
topdown	top-down segmentation images required for training LBC
2d_bbs_{front, left, right, rear}	2d bounding boxes for different agents in the corresponding camera view
3d_bbs	3d bounding boxes for different agents
affordances	different types of affordances
measurements	consists of the position, speed, and other metadata of the ego-agent.

In addition, the CARLA Leaderboard repository has a collection of scenarios as follow [64]:

Figure 31 Traffic scenario 01

Scenario 01: A loss of control that occurs without warning. Due to poor road conditions, the ego-vehicle loses control and must recover by returning to its original lane. [64]

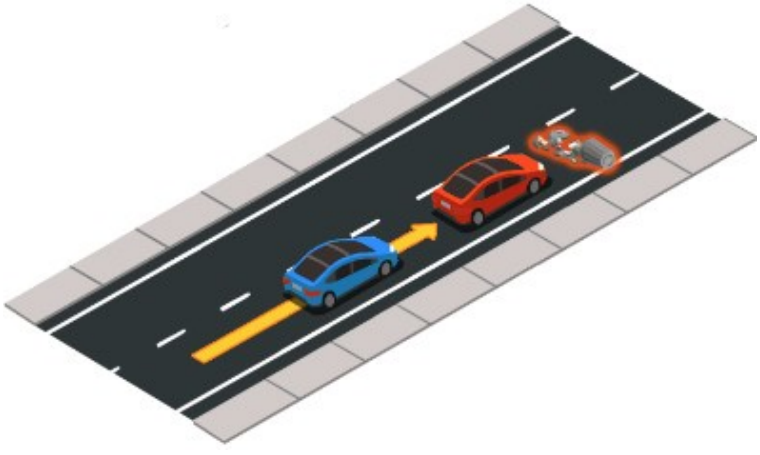


Figure 32 Traffic scenario 02

Scenario 02: After the leading vehicle's brake, longitudinal control. Due to an impediment, the vehicle in front of the go-vehicle must conduct an emergency stop or an avoidance maneuver.

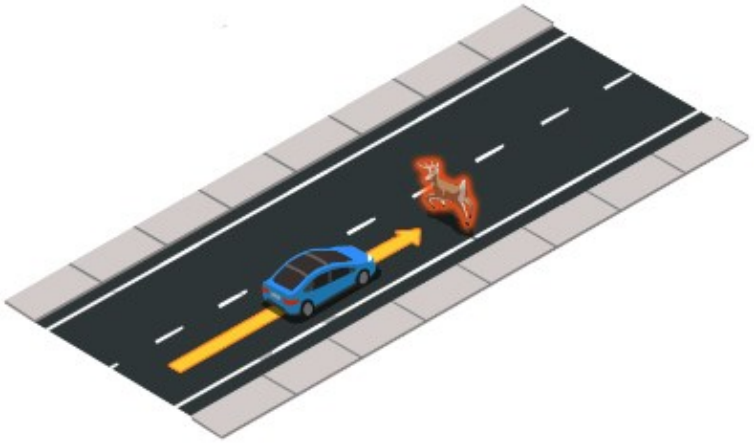


Figure 33 Traffic scenario 03

Scenario 03: Without prior activity, avoiding obstacles. On the road, the ego-vehicle encounters a barrier or unanticipated object and must conduct an emergency stop or evasive maneuver.

Figure 34 Traffic scenario 04

Scenario 04: Obstacle avoidance through proactive planning. During a maneuver, the ego-vehicle encounters a road obstruction and must apply the emergency brake or execute an avoidance maneuver.

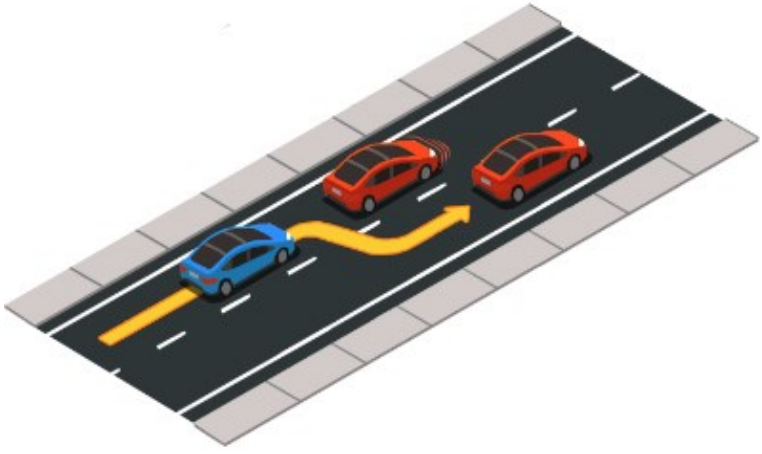


Figure 35 Traffic scenario 05

Scenario 05: Changing lanes to avoid a slow-moving leading car. The ego-vehicle changes lanes to avoid a car ahead that is travelling too slowly.

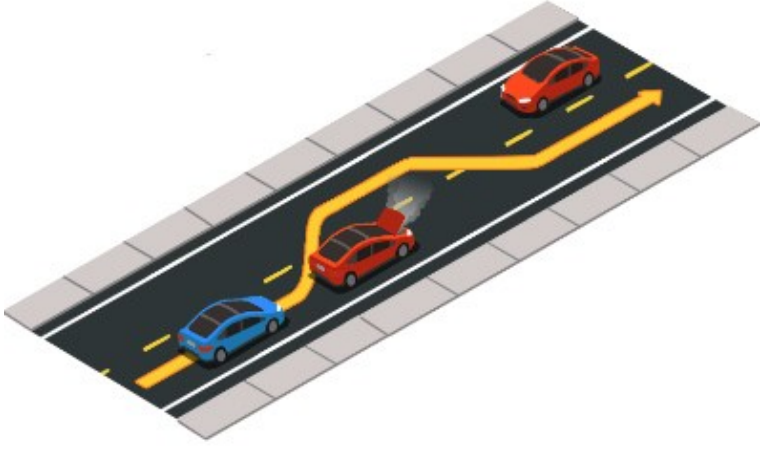


Figure 36 Traffic scenario 06

Scenario 06: Oncoming traffic is being handled by the passing vehicle. To avoid a barrier, the ego-vehicle must utilize the opposing lane and surrender to approaching traffic.

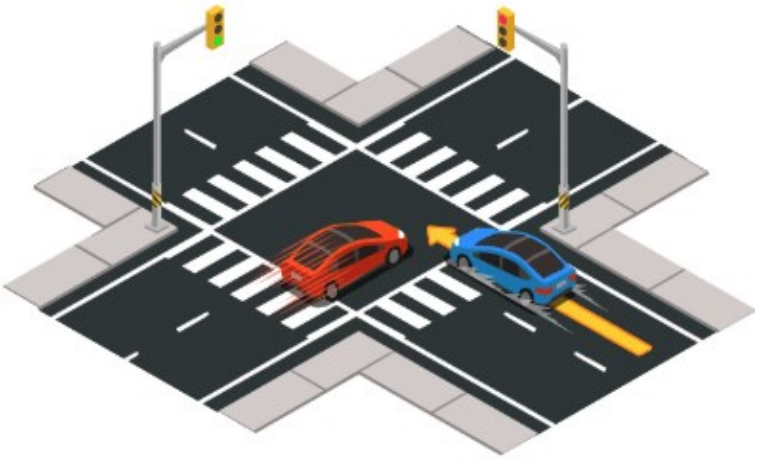


Figure 37 Traffic scenario 07

Scenario 07: Crossing traffic disobeys the red light at a junction. The ego-car is traveling straight through an intersection when a car crosses the street against the light and avoids a collision.

Figure 38 Traffic scenario 08

Scenario 08: Turn left unprotected at an intersection with approaching cars. At a crossroad, the ego-vehicle is making an unprotected left turn, yielding to oncoming traffic.

Figure 39 Traffic scenario 09

Scenario 09: Turning right at a crosswalk with moving traffic. At a crossroad, the ego-vehicle is making a right turn while giving way to oncoming traffic.

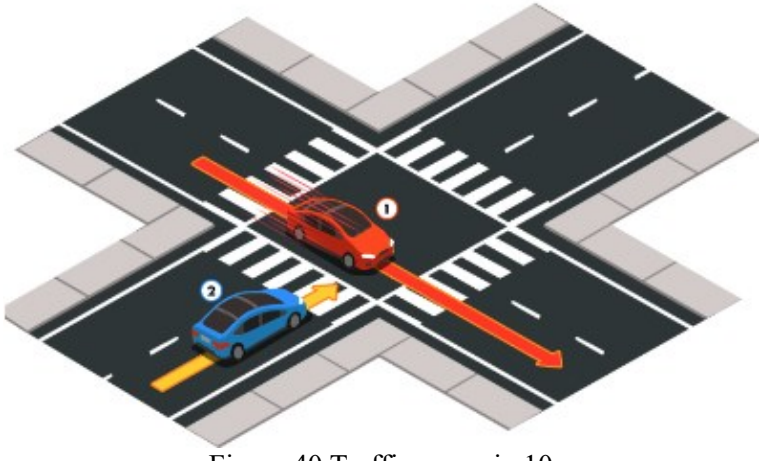


Figure 40 Traffic scenario 10

Traffic Scenario 10: Negotiating while crossing at an unsignalized junction. To cross an unsignalized junction, the ego-vehicle must bargain with other cars. In this case, it is deemed that whoever enters the junction first has precedence.

4.1. Training

In this section, we go through the process for the training CNN model.

Dataset: Data loading: Training a model involves traversing the dataset, taking a small batch of samples at a time, and using them to update our model. Since this process is the basis for training machine learning algorithms, it is necessary to define a function that disrupts the samples in the dataset and acquires the data in small batches.

Training: Initialize model parameters: Before we can start optimizing our model parameters with small batch stochastic gradient descent, we need to have some parameters first. Typically, the bias is set to zero and the weights are initialized by selecting random values from a normal distribution with a mean of 0 and a standard deviation of 0.01. In our case, we initialize the weights the same as our baseline model, which is defined in the config.py file. After initializing the parameters, our task is to update these parameters until they are sufficient to fit our data.

Next, we need to define the model (3.3 & 3.4), define the loss function in Eq. (14). And then we need to define the optimization algorithm in each step using a small randomly selected batch from the dataset, and then calculate the gradient of the loss based on the parameters. Finally, update our parameters in the direction of reducing the loss till it satisfies our condition. We choose the AdamW [65] as optimizer to make sure the training network can converge. We set our epoch number to 101, then enter the directory of the model to be trained and run the training script.

Training Metrics: We consider our predicted path planning as a regress type question; therefore, we can use the mean squared error and R2 score to evaluate the performance of the model.

Mean squared error is defined as below in Eq. (15), where n is the number of the samples, y_i is the real value, and $\hat{y}_i$ is the predicted value.

$$MSE\left(y,\hat{y}\right)=\frac{\sum_{i=1}^{n}\left(y_i-\hat{y}_i\right)^2}{n} \tag{15}$$

The MSE calculates the difference between each actual value and its corresponding projected value, squares those differences, adds them all together, and then divides the result by the total number of observations. For the regression model to be regarded as a decent model with good performance, the MSE should be as low as possible.

R2 Score can be defined as below in Eq. (16), where n is the number of the samples, y_i is the real value, $\hat{y}_i$ is the predicted value and $\overline{y}$ the average of real observed value.

$$R^2\left(y,\hat{y}\right)=1-\frac{\sum_{i=1}^{n}\left(y_i-\hat{y}_i\right)^2}{\sum_{i=1}^{n}(y_i-\overline{y})^2} \tag{16}$$

The R2 score is given as a value between zero and one. If the R2 score is zero, it means that our model is the same as the average and therefore our model needs to be improved. If the R2 score is one, the right-hand side of the Eq. (16) becomes zero, which only happens when our model fits each data point and there is no error. Hence, with the better model performance we want the R2 score is close to zero.

On the other hand, In the process of training a model, we also want to achieve two objectives.

1. The loss value of the training is as small as possible.
2. The difference between the trained loss value and the validated loss value is as small as possible.

Training Result: When we finish our training, we use tensorboard Visualization Framework to do the data analysis to verify the training model is qualified for the simulation evaluation test.

As we can see Fig. 41, the training loss function is converged at around 0.02, which is close to zero. And for Fig. 42, the validated loss is convergence.

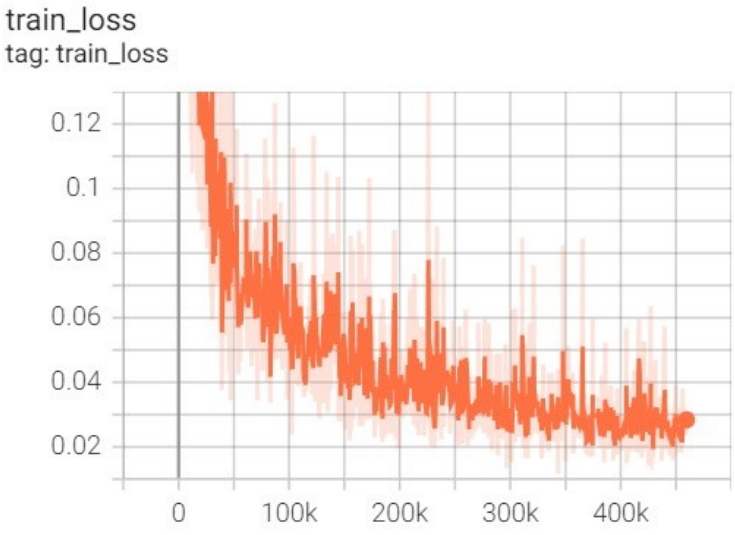


Figure 41 Train loss function on tensorboard

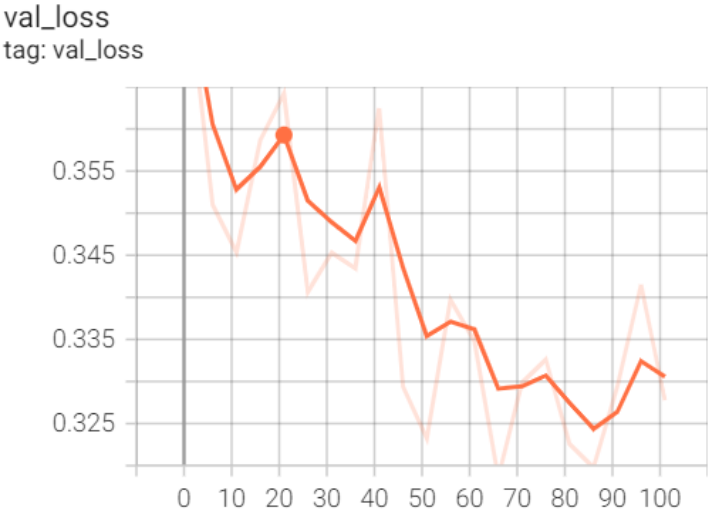


Figure 42 Validated loss function on tensorboard

The Fig. 43 and Fig. 44 show the training MSE converged around 0.01 more close to zero. The Fig 41 illustrates the validated MSE, we find the MSE value at the range between 80 and 90 epochs has the smallest value. Therefore, the best performance model should be within the range of 80 and 90 epochs. We verified the best model is the 86th model during the training. We were looking for the best training model from the log folder and found the best model is model_86.pth, which matches our analysis.

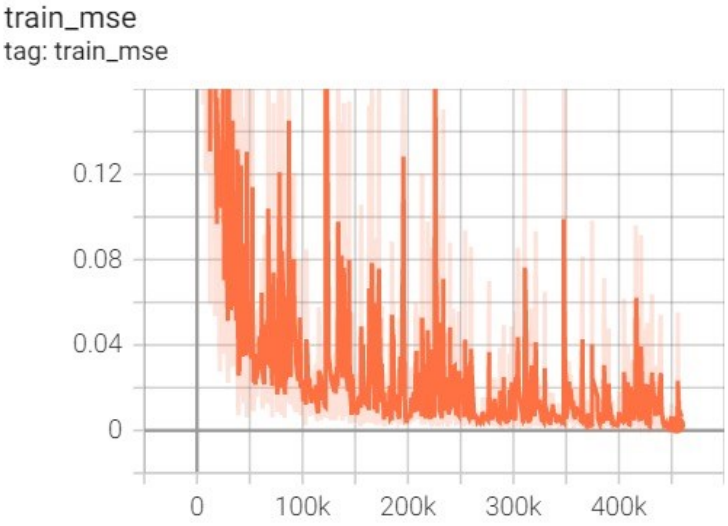


Figure 43 Training MSE on tensorboard

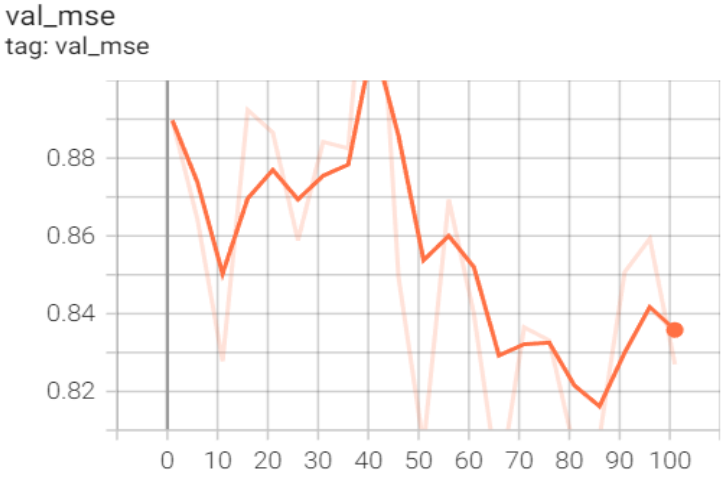


Figure 44 Validated MSE on tensorboard

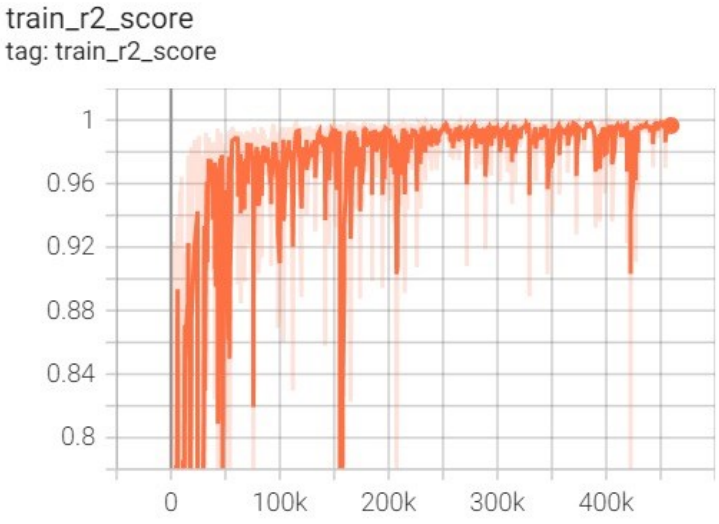


Figure 45 R2 score on tensorboard

The R^2 is around 0.98, which is closer to 1, the stronger explanation of y by the variables of the equation, the better the model fits the data.

When the purpose, which is that the loss value of the training is small, is not achieved, it means that the model is not trained well, and the model is not good at discriminating patterns or features of the data, then it is underfitting.

The reasons why the model is overfitting are shown:

1. The data in the training set is too small.
2. There is inconsistent feature distribution between the training set and the new data.
3. There is noise in the training set. The noise is so large that the model remembers the noisy features and instead ignores the true relationship between the inputs and outputs.
4. There are enough iterations of weight learning to fit the noise in the training data and the unrepresentative features in the training sample.

We can discard some features that do not help us predict correctly, and it can be a manual selection of which features to keep, or using some model selection algorithm to help, such as PCA.

4.2. Experimental Environment

We are using NVIDIA GeForce RTX 3090Ti for both training and simulation. Hence, we recommend using Conda to create the python3.7 virtual environment required for training and validation. Then, use *pip* to install the required dependencies. We use the CARLA 0.9.10 simulator for evaluating the model we trained on the map. Before training it should be noted that users should modify the model hyperparameters in *config.py* according to their own needs.

We need to make sure:

1. The *root_dir* in the config.py is the correct path for dataset folder.
2. The *train_towns* and *val_towns* are the datasets we want to use for training and validation.
3. The below parameters:

Table 5 The hyperparameter

learing rate	*lr*= 0.0001
image pre-processing	*crop*= 256
input timesteps	*seq_len* = 1
future waypoints predicted	*pred_len* = 4

Table 6 The GPT encoder parameter

n_embd	*block_exp*	*n_layer*	*n_head*
512	4	8	4
n_scale	*embd_pdrop*	*embd_pdrop*	*attn_pdrop*
4	0.1	0.1	0.1

We recommend using the default parameters in config.py. (Code link attached in Appendix A).

4.3. Simulation

After the model is trained, the log model folder will be generated in the folder where it is located, which contains the model parameter files *model.pth* and *best_model.pth* saved in each iteration during the model training process.

To verify the feasibility of the trained model, we simulated an autonomous driving scenario on CARLA simulator and used the leaderboard to interact. Follow the steps below to deploy and verify. First, we need to install and start CARLA server. Then we run the Autopilot, which launches the autopilot after the CARLA server is up and starts running to begin generating data. The detail of how to install the server can be found on Github. The link is provided in Appendix A.

4.4. Evaluation Results

We use Town05 Short as the simulation dataset to evaluate the performance of our proposed model under the complex scenario. Town05 Short data comprises 32 short routes, each between 100 and 500 meters long, with three intersections. The antagonistic situations and dynamic agents that make up each route are created at specific points along the route in high density.

Metric for Simulation: There are three official CARLA leaderboard metrics which are Route Completion, Score Penalty, and Driving Score. The definitions are below:

(1) Route Completion (RC): the average of the proportion of route distance R_i covered by the agent on route i over N routes. We represent it as Eq. (17).

$$RC = \frac{1}{N}\sum_{i}^{N} R_i \tag{17}$$

(2) Score Penalty: For each instance of violation j that the agent committed while on the route, the Infraction Multiplier (IM) calculates a geometric sequence of infraction penalty coefficients, or pj. An ideal base score of 1.0 is given to agents, and this value is decreased by a penalty coefficient for each transgression.

We represent it as Eq. (18)

$$IM = \prod_j (p_j)^{\#\ \text{inlraction}\ s_j} \tag{18}$$

(1) Driving Score (DS): route completion weighted average with infraction multiplier P. See Eq. (19)

$$DS = \frac{1}{N}\sum_i^N R_i P_i \tag{19}$$

In our case, we need to ensure that our design model we train has the best driving performance from the CARLA simulation test. Better driving performance means higher DS and RC with lower IM. We can also prove that our model has good fusion efficiency from this.

Model Complexity: Usually, when we evaluate a model, the first thing we should look at is its accuracy. Another aspect used to select the model is the complexity of an algorithm or model. The model needs more computational resources when the algorithm is more complex. Once our model has reached a certain level of accuracy, we need further evaluation metrics to evaluate our model: First, the computational power required for forwarding propagation, which reflects the level of performance required for hardware such as GPU; second, the number of parameters, which reflects the memory size occupied. Therefore, we introduce floating point operations ($FLOPs$), which indicate the number of floating point operations and the number of computation used to measure the complexity of an algorithm or model.

$$FLOPs = 2HW(C_{in}K^2 + 1)C_{out} \tag{20}$$

$$FLOPs = (2I - 1)O \tag{21}$$

Where H, W, and C_{in} are the height, width, and the number of channels of the input feature map,

K is the kernel width (assuming symmetry),

C_{out}out is the number of output channels [17].

The convolutional kernel uses Eq. (20) to calculate the $FLOPs$ and the fully connected layers use Eq. (21) to calculate $FLOPs$.

4.5. Result and Discussion

Baseline: We compared the PentaFusion model with Transfuser (baseline) [8] and other failed or successive versions of our designed models and selected the best performing model from them for improving the fusion efficiency. We designed four models, including Pentafusion_Epooling, Pentafusion_SE, EbFusion, and BiFusion.

EbFusion: Based on the idea of the Transfuser model, we add the channel attention SE-Net block on LiDAR and RGB after the feature extractor (T). Two feature maps will be concatenated before passing through the GPT block (2T). The output from the GPT block carrying the fusion information is divided into two parts, the first half will be added with the RGB feature map on the backbone, and the second half with the LiDAR feature map.

BiFusion: We apply the channel attention mechanism to the RGB branch to enhance global information. Besides that, we use the spatial attention mechanism on the LiDAR branch to increase the position information. The other part of BiFusion has the same operation as PentaFusion model in Fig. 25.

Pentafusion_Epooling: To solve the image information loss issue, when the pooling layer is down-sampling, we tried to remove the pooling layer after extracting feature maps from ResNet in the Pentafusion_Epooling model. However, it reaches the limitation of graphical computing if we do not use the pooling layer. So, we try to adjust the pooling size instead of removing the pooling layer. For the PentaFusion model in Fig. 25, we down-sample the feature map size from $64 \times 64 \times C$ into $8 \times 8 \times C$ ($H \times W \times C$) for all four layers in both LiDAR and camera streams. We proposed to change the down-sampling size ($H \times W \times C$) of the four layers into $32 \times 32 \times C$, $32 \times 32 \times C$, $16 \times 16 \times C$, and $8 \times 8 \times C$ respectively. The rest of the operations of the Pentafusion_Epooling are the same with the PentaFusion in Fig.25.

Pentafusion_SE: The idea of Pentafusion_SE is that we are trying to fuse more feature maps' information from LiDAR and RGB after feature extracting. The only difference between PentaFusion and Pentafusion_SE is that we add original feature maps from both LiDAR (T) and RGB (T) with the five branches (T) fused information (3T) before the GPT block process. Hence, the fused information (3T) will be completely integrated. The Pentafusion_SE model will separate into three parts, the first part will be passed back to the RGB branch, the second part will be given back to the LiDAR branch and the Pentafusion_SE model will drop the last part automatically during the fusion stage.

By comparing the simulation driving performance of each model, we can determine the best model for LiDAR and camera fusion in our self-driving car. In other words, better model performance means a higher efficiency for the fusion.

Results: In our experiment, to better compare the performance of each model on the CARLA simulator, we used Town5 Short as the evaluation dataset. This map contains 31 routes with complex scenarios. In order to ensure the accuracy of the results, we ran each model on this map five times and took the average value as a result. Fig. 47 reports the result of driving performance that Town05 Short with large densities of dynamic agents and situations; we provide the mean across five repeats run on two metrics which are Route Completion (RC) and Driving Score (DS). Fig. 46 shows the score penalty of each model.

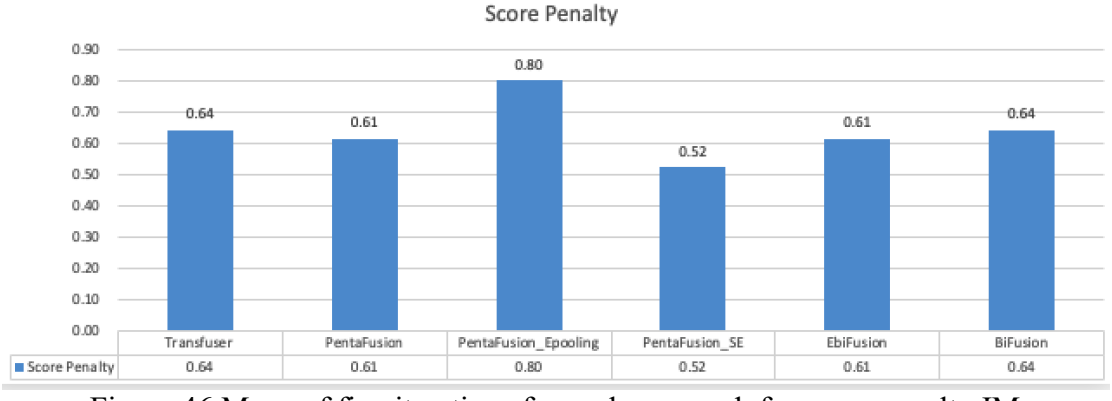


Figure 46 Mean of five iterations for each approach for score penalty IM

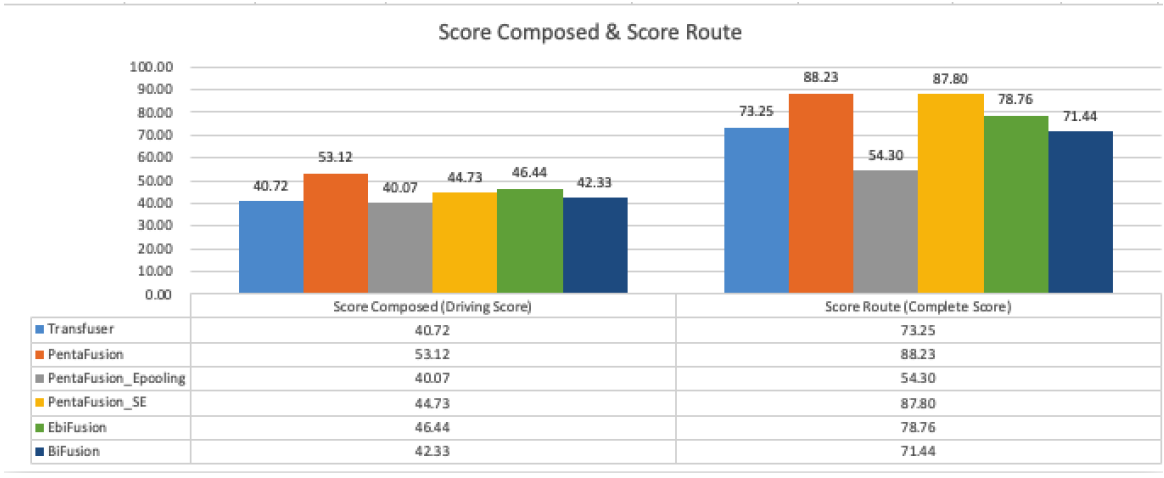


Figure 47 Mean of five iterations for each approach for two metrics: route completion (RC) and driving score (DS).

Transfuser is a strong baseline: As Fig. 46 and Fig. 47 show, The Transfuser DS is 40.72, RC is 73.25 and the penalty is 0.64. From Table 7 [8], we can see that compared to the performance with other methods. The CILRS is the camera-only method. Therefore, by comparing the driving performance, LiDAR and camera fusion's driving performance $DS = 40.72$ is much higher than the camera-only performance $DS = 7.47$. We prove that the sensor fusion is effort; the Transfuser fusion model has the best driving performance is shown in Table 7 [8].

Table 7 The driving performance of CILRS and LBS methods [3]

Method	**DS**	**RC**
Transfuser	40.72	73.25
CILRS (camera-only)	7.47	13.4
LBC	30.97	55.01

By comparing Transfuser's driving performance with BiFusion $DS = 42.33$ and EBfusion $DS = 46.33$ from Table. 8, we observe that the BiFusion model and EBfusion model have higher DS value than Transfuser, which illustrates that they have better performance than our

baseline Transfuser. Hence, we know that channel attention and spatial attention mechanisms are effective for fusing the LiDAR and camera information.

Next, we try to apply the spatial attention block and SE-block to both LiDAR and the image branches, which is the PentaFusion model. At this point, we observe the PentaFusion driving score of has increased $\frac{53.12-40.72}{40.72} = 30.45\%$ compared with Transfuser (baseline) and the route completion has increased $\frac{88.23-73.25}{73.25} = 20.45\%$.

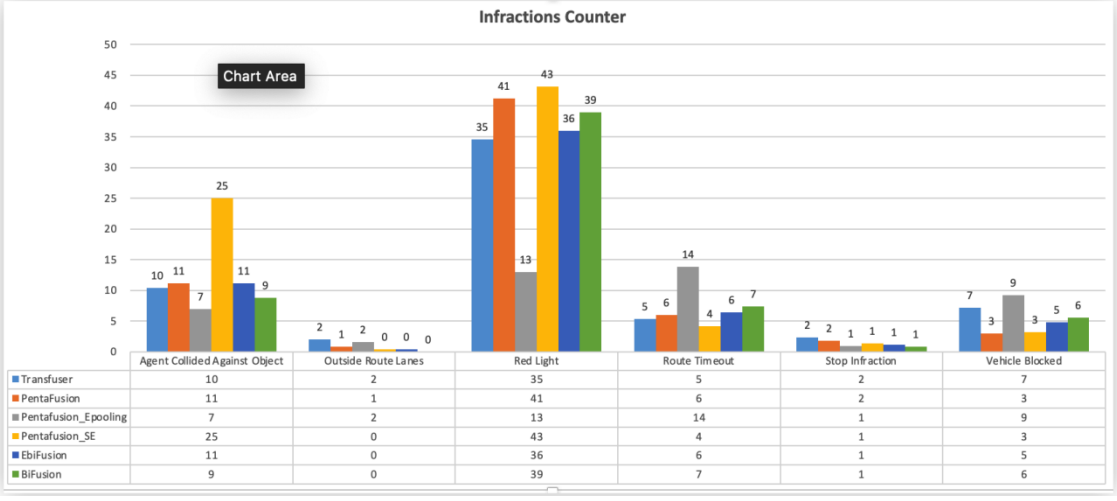
Table 8 Result of DS, RC, and IM for the methods we proposed

Method	**DS**	**RC**	**IM**
Transfuser (baseline)	40.72	73.25	0.64
BiFusion	42.33	71.44	0.64
EBfusion	46.44	55.01	0.61
Pentafuison (Best)	***53.12***	***88.23***	***0.61***
Pentafusion_Epooling	40.07	54.30	0.8
Pentafusion_SE	44.73	87.8	0.52

However, after analyzing the data of the infractions counter from Fig. 48, we find that all the models' performance is struggling with the red-light test. Due to the red-light signal being on the top or left front of the vehicle in the image, we think if the feature map loses some local information about the traffic light before fusion will lead the car to not recognize the light signal. Therefore, we consider if the invariance of the features is removed by pooling layer when doing compression after feature extraction. So, we decide to adjust the pooling layer to prevent that, and then we propose the design of Pentafusion_Epooling model. However, we observe the vehicle is stopped for no reason during the simulation while we adjust the down-sample feature map from size 8×8 (PentaFusion) to 32×32 (Pentafusion_Epooling). By analyzing the results from Table. 8, we find the route completion of Pentafusion_Epooling $RC = 54.30$ is worse than PentaFusion $RC = 88.23$. It proved the pooling layer can filter out some irrelative information under red-light situation. Therefore, the fusion model with the suitable size of pooling layer has a higher efficiency of information fusion.

As we can see, the result from PentaFusion has the highest driving score and the route completion for the score, which means the PentaFusion model has the best driving performed to pass the following tests: Agent Collided Against Object test, Outside Route Lanes test, Red Light test, Route Timeout test, Stop Infraction test, and Vehicle Blocked test. Fig. 48 shows the details of the number of infractions. Therefore, the PentaFusion model improves the efficiency and accuracy of context fusion in an end-to-end self-driving vehicle.

However, after examining the infractions counter data from Fig. 48, we discover that the PentaFusion model still has trouble passing the red-light test. Additionally, all other fusion models consistently run red lights.



	Agent Collided Against Object	Outside Route Lanes	Red Light	Route Timeout	Stop Infraction	Vehicle Blocked
Transfuser	10	2	35	5	2	7
PentaFusion	11	1	41	6	2	3
Pentafusion_Epooling	7	2	13	14	1	9
Pentafusion_SE	25	0	43	4	1	3
EbiFusion	11	0	36	6	1	5
BiFusion	9	0	39	7	1	6

Figure 48 Detail of the results: infractions counter

By comparing the FLOPs of each model from Table. 9, we find the FLOPs of the PentaFusion are $9{,}734.02M$, and the FLOPs of Transfuser is $11{,}397.85M$. This illustrates the PentaFusion model is faster than our baseline Transfuser. Due to the pooling layer, that will increase the number of calculations. The FLOPs of the Pentafusion_Epooling is $15{,}000.20M$, which is around 1.5 times bigger than the PentaFusion model. For the parameter size, the Pentafusion_Epooling is $9{,}734.24M$, the parameter size of Transfuser is $66.14M$, and the parameter size of PentaFusion is $73.81M$. We conclude that only the Pentafusion_Epooling

is significantly more extensive than other models. Moreover, the PentaFusion model's complexity is only slightly higher than the Transfuser model.

Table 9 The FLOPs and the parameter number for 6 methods

Model	**Input Resolution**	**Params （M）**	**FLOPS （M）**
Transfuser （Baseline）	256x256	66.14	11397.85
PentaFusion	256x256	73.81	9734.02
BiFusion	256x256	72.64	9666.78
Pentafusion_Epooling	256x256	9734.24	15000.20
Pentafusion_SE	256x256	73.90	9734.24
Ebfusion	256x256	72.76	9666.88

The data in Fig.46, Fig. 47, Fig.48, and Table 9 is the statistical raw experimental results of our simulation. We will provide all the original data of the experiment and put it on GitHub; the link is in Appendix A.

CHAPTER 5 CONCLUSION AND FUTURE WORK

In this paper, we comprehensively reviewed the deep learning-based sensor fusion approach by fusing LiDAR sensor and camera information at the feature extraction level. We output path planning point information and implement end-to-end autonomous driving by prediction points. We propose the trained-designed PentaFusion model to perform the fusion and validate it on CARLA simulation software. In the closed-loop experiments, we verify that our model's performance in complex scenarios improved over our baseline Transfuser model by increasing the average composite score from 40.72 to 53.12, which increased by 30.45%. The number of error infraction rate in complex scenarios, such as collision rate and vehicle block rate, was reduced. We also validated our proposed method by increasing the map completion rate on complex terrain by 15% over the transfuser. So, we can say that our proposed scheme for improving sensor fusion in the medium term makes sensor fusion work better.

We proposed BEV and camera front view for fusion, using ResNet18 and ResNet34 as the backbone networks, respectively, to do feature extraction, and then the extracted features are learned for spatial correlation and correlation between channels, respectively. They fused the feature map obtained for combining the learned weights with the feature map. The new fused feature map was able to carry more information.

From the analysis of the simulated experimental data of the model, it is shown that the new fusion method improves the effective fusion efficiency of location information by a low accident rate of self-driving cars in complex situations such as traffic lights and multiple vehicles. For the image information in the input to the network, after layer-by-layer convolution, the semantic information will become more and more obvious, and the relative location information will become weaker. The higher the level of convolution, the larger the perceptual field of the feature map mapped to the original map; thus, the perception of local location information is poorer. Our proposed spatial attention model can solve this problem well. And the self-attention mechanism increases the fusion of temporal features so that the fused information can contain the effective information features immediately. Then we

compared the fusion mechanism of self-attention and cross-attention and demonstrated the effectiveness of self-attention in the application of autonomous driving scenarios. In addition, we compared Convnet as the backbone network, which is not as effective as ResNet in feature extraction in our application scenario. After the feature extraction, we increased the value of the original down sample. We found that the test completion in the simulation experiment was lower. We observed that the self-driving car stopped running at some moments for no reason. This was due to the performance of the model becoming worse after reducing pooling size. We can conclude that the down-sampling process filters out some useless feature information to prevent overfitting; Thus, the model works better when the feature map is compressed to $8 \times 8 \times 64$. We compared the efficiency of the proposed PentaFusion scheme by applying the attention mechanism to different locations. We conclude the PentaFusion model has the best driving performance than others.

Based on our analysis and experimentation, we propose three directions for future research:

(1) Extending the PentaFusion model to additional sensors such as radar, due to radar having similar 3-D point cloud sensor data.

(2) Use Rangeview LiDAR information. We can also try to replace the BEV to provide more context for fusion.

(3) Extending the PentaFusion model to other dataset maps under different climate scenarios.

BIBLIOGRAPHY

[1] J. Shuttleworth, "SAE J3016 automated-driving graphic," *www.sae.org*, Jan. 07, 2019. https://www.sae.org/site/news/2019/01/sae-updates-j3016-automated-driving-graphic (accessed Jul. 30, 2022).

[2] A. Filos, P. Tigas, R. McAllister, N. Rhinehart, S. Levine, and Y. Gal, "Can autonomous vehicles identify, recover from, and adapt to distribution shifts?," *PMLR*, pp. 3145--3153, Sep. 2020, Accessed: Jul. 30, 2022. [Online]. Available: https://arxiv.org/abs/2006.14911.

[3] J. Kocic, N. Jovicic, and V. Drndarevic, "Sensors and sensor fusion in autonomous vehicles," *2018 26th Telecommunications Forum (TELFOR)*, pp. 420--425, Nov. 2018.

[4] D. Feng *et al.*, "Deep multi-modal object detection and semantic segmentation for autonomous driving: datasets, methods, and challenges," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–20, 2020.

[5] V. Vielzeuf, A. Lechervy, S. Pateux, and F. Jurie, "Multilevel sensor fusion with deep learning," *IEEE Sensors Letters*, vol. 3, no. 1, pp. 1–4, Jan. 2019.

[6] J. Fayyad, M. A. Jaradat, D. Gruyer, and H. Najjaran, "Deep learning sensor fusion for autonomous vehicle perception and Localization: a review," *Sensors*, vol. 20, no. 15, p. 4220, Jul. 2020.

[7] L. Cultrera, L. Seidenari, F. Becattini, P. Pala, and A. Del Bimbo, "Explaining autonomous driving by learning end-to-end visual attention," *openaccess.thecvf.com*, pp. 340–341, 2020, Accessed: Jul. 30, 2022. [Online]. Available: https://openaccess.thecvf.com/content_CVPRW_2020/html/w20/Cultrera_Explaining_Autonomous_Driving_by_Learning_End-to-End_Visual_Attention_CVPRW_2020_paper.html.

[8] A. Prakash, K. Chitta, and A. Geiger, "Multi-modal fusion transformer for end-to-end autonomous driving," *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7077--7087, 2021.

[9] Y. Zhang, H. Liu, and Q. Hu, "Transfuse: Fusing transformers and cnns for medical image segmentation," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*, Springer, 2021, pp. 14--24. Accessed: Jul. 30, 2022. [Online]. Available: https://arxiv.org/abs/2102.08005.

[10] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, "A convnet for the 2020s," *arXiv:2201.03545 [cs]*, pp. 11976--11986, Jan. 2022, [Online]. Available: https://arxiv.org/abs/2201.03545.

[11] R. Glon, “Tesla admits its Full Self-Driving technology is a Level 2 system,” *Autoblog*, Mar. 09, 2021. https://www.autoblog.com/2021/03/09/tesla-full-self-driving-level-2-sae/.

[12] “Self-driving cars and trucks market size | industry report, 2020-2030,” *Grandviewresearch.com*, 2018. https://www.grandviewresearch.com/industry-analysis/driverless-cars-market.

[13] Tesla, “Autopilot,” *Tesla.com*, 2019. https://www.tesla.com/autopilot (accessed 2018).

[14] “Learn how waymo drives - waymo help,” *support.google.com*, 2022. https://support.google.com/waymo/answer/9190838?hl=en.

[15] A. Zhang, Z. Lipton, M. Li, and A. Smola, “Dive into deep learning,” *arXiv preprint arXiv:2106.11342*, 2021, [Online]. Available: https://zh.d2l.ai/d2l-zh.pdf.

[16] F.-F. Li, “CS231n convolutional neural networks for visual recognition,” *Github.io*, 2012. https://cs231n.github.io/convolutional-networks/.

[17] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, “Pruning convolutional neural networks for resource efficient inference,” *arXiv preprint arXiv:1611.06440*, 2016, Accessed: Aug. 01, 2022. [Online]. Available: https://arxiv.org/pdf/1611.06440v2.pdf.

[18] B. Bai, “CNN,” *cnds*. May 20, 2020. https://www.woshipm.com/ai/3884563.html.

[19] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” arXiv preprint arXiv:1409.1556, 2014.

[20] Irhum Shafkat, “Intuitively Understanding Convolutions for Deep Learning,” *Medium*, Jun. 2018. https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1.

[21] M. Ge, “CNN_pooling_layer,” *csdn*. https://blog.csdn.net/weixin_44014740/article/details/110068805?spm=1001.2014.3001.5502.

[22] N. Akhtar and U. Ragavendran, “Interpretation of intelligence in CNN-pooling processes: a methodological survey,” *Neural Computing and Applications*, Jul. 2019.

[23] V. Pattabiraman and H. Singh, “Deep learning based brain tumour segmentation,” *IETE Journal of Research*, vol. 19, pp. 1--9, Jan. 2021.

[24] F.Li, “[CS231n] 6. Training neural networks I,” *Fresh-Learning*, Feb. 26, 2019. https://leechamin.tistory.com/96 (accessed Aug. 01, 2022).

[25] M. Ge, “CNN activation layer,” *cnds*. https://blog.csdn.net/weixin_44014740/article/details/110048444.

[26] J. Goutsias, L. M. Vincent, D. S. Bloomberg, and I. Netlibrary, *Mathematical morphology and its applications to image and signal processing*, vol. 18. Boston: Springer Science \& Business Media, 2000.

[27] Y. Cui *et al.*, "Deep learning for image and point cloud fusion in autonomous driving: a review," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 2, pp. 722–739, Feb. 2022.

[28] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, May 2012.

[29] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *Very deep convolutional networks for large-scale image recognition*, 2014.

[30] C. Szegedy *et al.*, "Going deeper with convolutions," *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1--9, 2015, [Online]. Available: https://arxiv.org/abs/1409.4842.

[31] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *International conference on machine learning*, pp. 448--456, 2015.

[32] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818--2826, 2016, [Online]. Available: https://arxiv.org/abs/1512.00567.

[33] C. R. Qi, W. Liu, C. Wu, H. Su, and Guibas, Leonidas J, "Frustum pointnets for 3d object detection from rgb-d data," *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 918--927, 2018, Accessed: Dec. 18, 2019. [Online]. Available: https://arxiv.org/abs/1711.08488.

[34] Z. Yang, Y. Sun, S. Liu, X. Shen, J. Jia, and Y. Lab, "Ipod: Intensive point-based object detector for point cloud," *arXiv preprint arXiv:1812.05276*, 2018, Accessed: Aug. 01, 2022. [Online]. Available: https://arxiv.org/pdf/1812.05276v1.pdf.

[35] X. Zhao, Z. Liu, R. Hu, and K. Huang, "3D object detection using scale invariant and feature reweighting networks," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, pp. 9267–9274, Jul. 2019.

[36] W. Chen, P. Li, and H. Zhao, *MSL3D: 3D object detection from monocular, stereo and point cloud for autonomous driving*, vol. 494. Elsevier, 2022, pp. 23–32.

[37] J. Ku, M. Mozifian, J. Lee, A. Harakeh, and S. Waslander, "oint 3d proposal generation and object detection from view aggregation," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2018, pp. 1--8. Accessed: Sep. 14, 2021. [Online]. Available: https://arxiv.org/abs/1712.02294.

[38] M. Liang, B. Yang, S. Wang, and R. Urtasun, "Deep continuous fusion for multi-sensor 3d object detection," Proceedings of the European conference on computer vision (ECCV), 2018.

[39] B. Li, T. Zhang, and T. Xia, "Vehicle detection from 3d lidar using fully convolutional network," *arXiv preprint arXiv:1608.07916*, Aug. 2016, Accessed: Aug. 01, 2022. [Online]. Available: https://arxiv.org/abs/1608.07916.

[40] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-view 3d object detection network for autonomous driving," *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 1907--1915, Jun. 2017.

[41] Y. Zhou and O. Tuzel, "Voxelnet: End-to-end learning for point cloud based 3d object detection," *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4490--4499, Nov. 2017.

[42] C. R. Qi, L. Yi, H. Su, and Guibas, Leonidas J, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," *Advances in neural information processing systems*, vol. 30, 2017.

[43] B. Nojavanasghari, D. Gopinath, J. Koushik, T. Baltrušaitis, and L.-P. Morency, "Deep multimodal fusion for persuasiveness prediction," *Proceedings of the 18th ACM International Conference on Multimodal Interaction*, pp. 284--288, Oct. 2016.

[44] H. Wang, A. Meghawat, L.-P. Morency, and E. P. Xing, "Select-additive learning: Improving generalization in multimodal sentiment analysis," *IEEE*, pp. 949--954, Sep. 2016.

[45] A. Anastasopoulos, S. Kumar, and H. Liao, "Neural language modeling with visual features," *arXiv preprint arXiv:1903.02930*, Mar. 2019, Accessed: Aug. 01, 2022. [Online]. Available: https://arxiv.org/abs/1903.02930.

[46] V. Vielzeuf, A. Lechervy, S. Pateux, and F. Jurie, "Centralnet: a multilayer approach for multimodal fusion," *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, Aug. 2018.

[47] B. Zhou, Y. Tian, S. Sukhbaatar, A. Szlam, and R. Fergus, "Simple baseline for visual question answering," *arXiv preprint arXiv:1512.02167*, Dec. 2015.

[48] J.-M. Pérez-Rúa, V. Vielzeuf, S. Pateux, M. Baccouche, and F. Jurie, "Mfas: Multimodal fusion architecture search," *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6966--6975, 2019.

[49] D. Bahdanau, K. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” *arXiv preprint arXiv:1409.0473*, 2014.

[50] A. Graves, G. Wayne, and I. Danihelka, “Neural turing machines,” *arXiv preprint arXiv:1410.5401*, Dec. 2014.

[51] A. Vaswani *et al.*, “Attention is all you need,” *Advances in neural information processing systems*, vol. 30, Dec. 2017.

[52] C. Zhang, Z. Yang, X. He, and L. Deng, “Multimodal intelligence: Representation learning, information fusion, and applications,” *IEEE Journal of Selected Topics in Signal Processing*, vol. 14, no. 3, pp. 478–493, Mar. 2020.

[53] C.-F. Chen, Q. Fan, and R. Panda, “Crossvit: Cross-attention multi-scale vision transformer for image classification,” *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 357--366, Aug. 2021.

[54] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770--778, Dec. 2015.

[55] K. Chen, K. Chen, Q. Wang, Z. He, J. Hu, and J. He, “Short-Term load forecasting with deep residual networks,” *IEEE Transactions on Smart Grid*, vol. 10, no. 4, pp. 3943–3952, Jul. 2019.

[56] A. Prakash, K. Chitta, and A. Geiger, “Multi-modal fusion transformer for end-to-end autonomous driving,” *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7077--7087, 2021.

[57] N. Rhinehart, R. Mcallister, K. Kitani, and S. Levine, “Precog: Prediction conditioned on goals in visual multi-agent settings,” 2019.

[58] A. Prakash, K. Chitta, and A. Geiger, “Supplementary material for multi-modal fusion transformer for end-to-end autonomous driving,” 2021.

[59] Y. Zhang, H. Liu, and Q. Hu, “Transfuse: Fusing transformers and cnns for medical image segmentation,” in *International Conference on Medical Image Computing and Computer-Assisted Intervention*, Springer, 2021, pp. 14–24.

[60] Hu, L. Shen and G. Sun, "Squeeze-and-Excitation Networks," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 7132-7141.

[61] S. Woo, J. Park, J.-Y. Lee, and I. Kweon, “Cbam: Convolutional block attention module,” *Proceedings of the European conference on computer vision (ECCV)*, pp. 3--19, [Online]. Available: https://arxiv.org/pdf/1807.06521.pdf.

[62] S. Zagoruyko and N. Komodakis, “Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer,” *arXiv preprint arXiv:1612.03928*, 2016, [Online]. Available: https://arxiv.org/pdf/1612.03928.pdf.

[63] K. Cho, B. Van Merriënboer, C. Gulcehre, F. Bougares, H. Schwenk, and Y. Bengio, “Learning phrase representations using RNN encoder-decoder for statistical machine translation,” *arXiv preprint arXiv:1406.1078*, 2014.

[64] “Traffic Scenarios,” *CARLA Autonomous Driving Leaderboard*. https://leaderboard.carla.org/scenarios/ (accessed Aug. 01, 2022).

[65] “Adamw and super-convergence is now the fastest way to train neural nets,” *last accessed*, vol. 19, 2018.

[66] “The GPT-3 architecture,” *dugas.ch*. https://dugas.ch/artificial_curiosity/GPT_architecture.html (accessed Aug. 01, 2022).

APPENDIX A Link of Scripts for Experimentations

We provide the links for our code on GitHub, which contains all our experimentations code, training results and simulated evaluation results, as well as the models that were mentioned in the experimentation chapter (Chapter 5).

1. GitHub repository: https://github.com/zhangjenny/AutoDrive.