Hochschule Karlsruhe
University of
Applied Sciences

3D Object Detection for Autonomous Driving

Projekarbeit

Implementation of VoxelNext model with nuScenes and KITTI Dataset.

Student:	Lucía Montero Couso, Bibek Gupta, Luis Sánchez Fernández
University:	Hochschule Karlsruhe – Technik und Wirtschaft
Course of study:	Mechanics and mechatronics
Studienvertiefung:	EU4M
Semester:	Winter semester 2023
Dozent:	Prof. Dr.-Ing. Stefan Trittler
Edited on:	31. March 2024

Contents

1 Abstract

In this project we collaborated with an existing project available on GitHub to address the challenges associated with 3D object detection. The project has an Apache licence, that allow us use, modify and distribute the software,, including creating derivative works, without requiring those derivative works to be licensed under the same terms. Rather than creating a new solution from scratch, the tool called VoxelNext was leveraged, as it was already developed and accessible on GitHub. [1]

In this study, we thoroughly examine how VoxelNeXt's architecture is adapted to suit different datasets, particularly the nuScenes and KITTI datasets. We break down the various components of VoxelNeXt and discuss the adjustments made to make it compatible with these datasets. Through our experiments and analysis, we aim to understand how well VoxelNeXt performs and the challenges it faces in real-world scenarios.

By delving into the specifics of VoxelNeXt's structure and its customization process, we explore its functionality and effectiveness. We conduct extensive tests and evaluations to accurately assess its performance. Our objective is to identify any obstacles encountered and the strategies implemented to address them, providing valuable insights into the model's behavior when applied to practical scenarios.

Ultimately, this report aims to offer a comprehensive understanding of how VoxelNeXt is tailored to accommodate different datasets, offering valuable insights for future research and development in the domain of 3D object detection.

2 Introduction

The evolution of artificial vision and sensor technology has been remarkable since its origin in the 19th century. It all started with the invention of cameras and the exploration of scientific imaging methods. Over time, advancements led to breakthroughs like telescopic images and radiographs.

It was not until the mid-20th century that the industrial artificial vision emerged. This was possible by early prototypes of vision cameras and image processing systems, which used computers and software to analyse captured images. The real turning point came in the 1980s when significant technological leaps occurred. Improved computing power allowed microprocessors to capture, process, and reproduce images from remote cameras. This marked the beginning of automated vision processes, where software could interpret image data to replicate visual characteristics.

Sensor technology has also progressed significantly, from early mechanical devices to modern electronic sensors like CCDs and CMOS. These sensors offer enhanced sensitivity and resolution, enabling more precise detection and tracking of objects in three-dimensional environments.Today artificial vision, as applied in projects like "Fully Sparse VoxelNet for 3D Object Detection and Tracking," benefits greatly from advancements in artificial intelligence and machine learning. The integration of machine learning algorithms and neural networks enhances the analysis, evaluation, and response capabilities of artificial vision systems, allowing them to continuously adapt and improve based on accumulated data and experience.

Today, artificial vision has evolved alongside advancements in artificial intelligence and machine learning. By integrating these technologies, vision systems can continuously learn and improve, making them invaluable in various industries.

In the domain of autonomous driving, achieving accurate 3D perception is crucial to ensuring safe navigation. Traditional methods of 3D object detection have historically depended on manually crafted features and dense point anchors. However, these approaches often struggle to effectively capture the sparsity and irregularity inherent in 3D data. VoxelNeXt offers a novel approach by directly predicting 3D objects from voxel features using a fully sparse convolutional network, complemented by CNN backbone adaptation. This fusion not only simplifies the detection pipeline but also significantly enhances inference efficiency, making it a promising solution for 3D object detection and tracking in autonomous driving scenarios.

VoxelNeXt combines the power of voxel-based feature extraction with the adaptability of convolutional neural network (CNN) backbones. Unlike its predecessors, VoxelNeXt leverages CNN backbone adaptation to tailor the feature extraction process specifically for 3D object detection tasks. By seamlessly integrating CNN backbones into the architecture, VoxelNeXt achieves a balance between robust feature representation and computational efficiency, ensuring accurate

detection even in sparse and irregular 3D environments.

This innovative approach not only eliminates the need for complex conversions but also enables VoxelNeXt to effectively capture fine-grained geometric information, essential for precise object detection. By embracing sparse convolutional networks and CNN backbone adaptation, VoxelNeXt stands at the forefront of advancements in autonomous driving technology, offering a scalable and efficient solution for navigating complex environments with confidence and accuracy.

The evolution of sensor technology, particularly Light Detection and Ranging (LiDAR), has played a pivotal role in enhancing the capabilities of 3D perception systems like VoxelNeXt. LiDAR sensors, which emit laser pulses to measure distances and create detailed 3D point clouds of surroundings, provide crucial data for accurately detecting and tracking objects in autonomous driving scenarios. By incorporating LiDAR data alongside voxel features and CNN backbone adaptation, VoxelNeXt can effectively leverage the rich spatial information provided by LiDAR sensors to enhance its object detection and tracking capabilities even further.

2.1 Autonomous vehicles (AVs)

AVs are vehicles equipped with fully automated driving systems capable of navigating roads without human intervention. Key modules within AV systems include sensing, environment perception, mapping and localization, prediction, planning, and control, which will be discussed further below.

2.1.1 Functional Modules of Autonomous Driving Systems

To enable vehicles to drive autonomously and reduce human errors, they incorporate several vital components:

- **Sensing:** Uses several sensors to understand the vehicle's status and its surroundings.

- **Perception:** Builds a contextual understanding of the environment by analyzing sensor data. This includes identifying and tracking nearby objects, detecting road signs, lanes and other vehicles' speeds.

- **Mapping and Localization:** Creates a map of the surroundings and determines the vehicle's precise location within it. This information helps the vehicle navigate roads, lanes, and follow traffic rules.

- **Prediction:** Anticipates the behaviors and trajectories of nearby objects, aiding in recognizing potential hazards and reducing errors in understanding the environment.

- **Planning:** Makes decisions about the vehicle's path and actions based on the gathered information. This involves planning routes, behaviors, and maneuvers to ensure safe and efficient travel.

- **Control:** Executes the planned actions by controlling vehicle functions such as steering, braking, signaling, and lighting. Precise control mechanisms minimize errors in executing driving commands.

In the context of this study, emphasis will be placed on the first two components, where the development will focus on training a program to effectively detect and identify various objects that could potentially be encountered within proximity.

2.1.2 Autonomous Vehicle Sensors

Sensor technology enables vehicles to perceive and understand their environment. These sensors play a critical role in providing essential data for the other modules.

Sensing is a critical aspect of fully automated driving systems, relying heavily on inputs from various sensors. These sensors can be broadly categorized into three main types:

- **Self-sensing:** Sensors such as odometers, gyroscopes, inertial measurement units (IMUs), and the control area network (CAN) bus constantly compute the vehicle's state, including velocity, acceleration, and steering angle.

- **Localization:** Sensors such as the Global Positioning System (GPS) determine the vehicle's local and global positions, assisting in navigation and route planning.

- **Surrounding Sensing:** This involves capturing information about the surrounding driving environment to enable the vehicle to "perceive" the road, lanes, objects, and traffic signs.

Among the various sensor modalities used in autonomous vehicles, three are commonly employed for perception tasks:

- **Cameras:** Monocular cameras provide detailed information about objects within their field of view but lack depth information crucial for accurate position estimation. Stereo cameras use multiple lenses to recover depth information, while RGB-D cameras offer synchronized depth and color within their field of view.

- **LiDARs:** LiDAR has become important in the perception of these vehicles because it can accurately measure depth using laser light. This generates a 3D Point Cloud that can be

analysed to detect objects around the vehicle. But using LiDAR data has its problems, especially with point density varying with distance. In popular datasets like Kitty, it has been observed that distant objects have fewer points due to the LiDAR's point pattern. Most methods work around point density variations by using voxel-based encoding that lose fine-grained geometric information, such as in this project. This method ignores point density as an additional encoding feature but in order to address this, sparse-to-dense conversion techniques can be employed. This method fills in the gaps in sparse point clouds, creating a denser representation for more accurate analysis.

- **Radars:** Radar sensors transmit and receive radio waves to determine the angle, range, and velocity of objects, particularly useful for detecting objects in mid to long-range distances. However, they may struggle to detect stationary objects.

While all sensor types contribute to the perception capabilities of autonomous vehicles, this project will primarily focus on LiDAR data for its role in facilitating 3D object detection. However, it's essential to address the challenges associated with LiDAR data, particularly point density variations, which can affect the accuracy of object detection.

2.1.3 LiDAR-only Object Detection

In this section, several methods are proposed for 3D object detection, with a focus on LiDAR point clouds as the primary input sensor modality. These methods employ different strategies to address the unstructured and sparse nature of LiDAR point clouds.

- **Point-cloud processing Methods:**PointNet is a method used for analyzing raw point-cloud data, like those from LiDAR sensors, for tasks such as object classification and segmentation. Unlike other methods, it does not need to convert the point-cloud data into a structured format first. Instead, it directly works with the original unstructured data. To handle the fact that the order and orientation of points in the cloud can vary, PointNet uses a simple but powerful technique called maxpooling.

  PointNet is great because it is robust. It can handle outliers and missing data well. However, it was initially tested on smaller datasets with around 1,000 points, while real-world LiDAR scenes in things like autonomous driving can have around 100,000 points.

  PointNet++ was introduced to improve upon PointNet's ability to capture local structures within the point-cloud data. PointNet++ breaks down the point-cloud into smaller overlapping parts and applies PointNet to each part individually. This way, it can better understand the hierarchical structure of the data.

- **Volume-based methods:** represent a significant advancement in the realm of 3D object detection using LiDAR point clouds. These methods address the challenge of working with unstructured and sparse LiDAR data by partitioning the 3D space into smaller units for analysis.

VoxelNet discretizes the 3D space into equally spaced cubic volumes called voxels. Each voxel either contains LiDAR points or remains empty. These voxels are then processed through a series of layers called Voxel Feature Encoding (VFE) layers, which encode the features of each non-empty voxel hierarchically. These encoded voxel-wise features then pass through a sequence of 3D convolutional layers. The output of this convolutional process is fed into a Region Proposal Network (RPN) for further analysis. VoxelNet predicts object targets from a Bird's Eye View (BEV) perspective, simplifying object detection as all objects lie on the same ground plane. The model is trained on datasets like KITTI, with data augmentation techniques applied to mitigate overfitting. Despite its superior accuracy, VoxelNet's computational complexity is high due to the use of 3D convolutions.

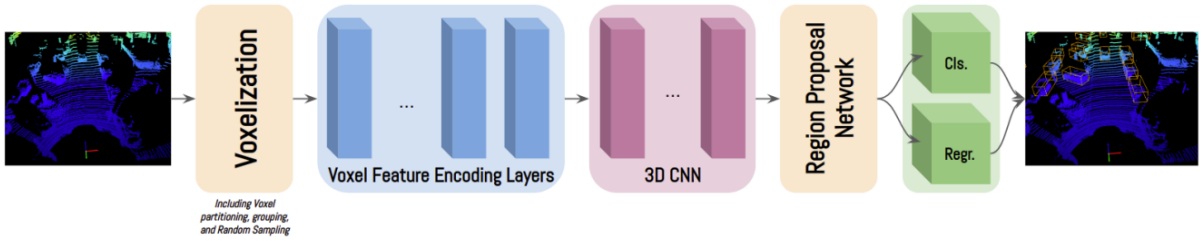


Figure 1: Architecture overview for VoxelNet [6]

PointPillars presents an alternative approach to VoxelNet's voxel-based method. Instead of voxels, PointPillars divides the 3D space into vertical columns called pillars along the x-y plane.

In contrast, PointPillars adopts a different approach by organizing the LiDAR point cloud into pillar structures within the x-y plane instead of using 3D voxels. These pillars capture the spatial information of the LiDAR point-cloud in a structured manner, facilitating easier processing with 2D convolutions. PointPillars employs a PointNet architecture to learn the encoding of each pillar. The pillar-wise encodings are then combined to form a 2D pseudo-image, which is processed by a 2D CNN backbone followed by a Single Shot Detector (SSD)-like network to detect objects. Similar to VoxelNet, PointPillars also uses predefined anchors and applies data augmentation techniques for robust training. Notably, PointPillars achieves state-of-the-art performance on benchmarks like KITTI while running at a significantly faster inference speed compared to other methods.

- **Projection-based methods:** they have been very important in the development of efficient object detection algorithms for LiDAR point-cloud data, particularly the Bird's Eye View (BEV) projection.

  PIXOR (Pixel-wise Occupancy with Recursive Neural Networks) is a single-stage 3D object detector that operates by representing the LiDAR point-cloud from a top-view

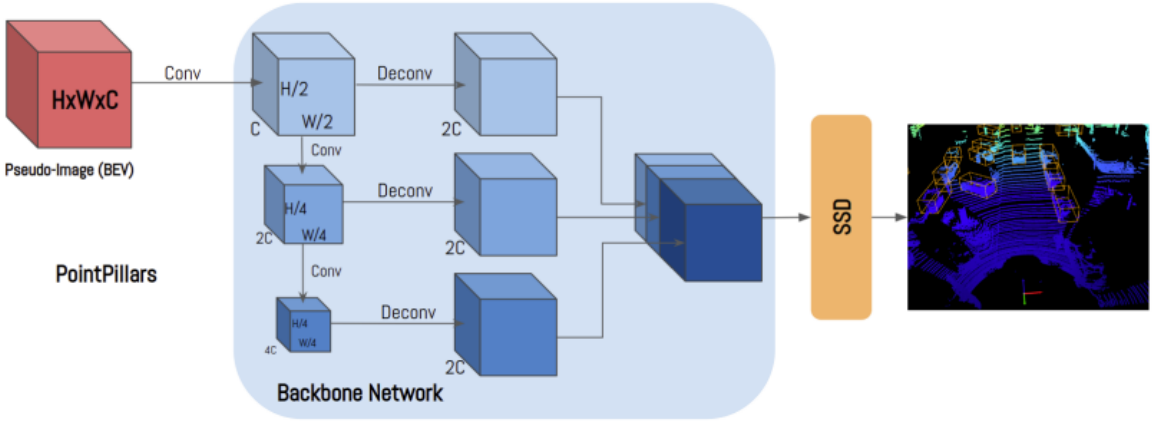


Figure 2: Backbone network architecture overview for PointPillars [6]

perspective, creating a 3D occupancy grid. This grid discretizes the physical space into small cells, indicating the presence of LiDAR points within each cell. PIXOR uses a fully-convolutional architecture for end-to-end learning, with a backbone network for feature extraction and a header network for generating detections. It employs techniques like focal loss for objectness prediction and smooth L1 loss for regression. PIXOR achieves high detection accuracy on the KITTI benchmark while maintaining real-time performance, outperforming other BEV-based methods.

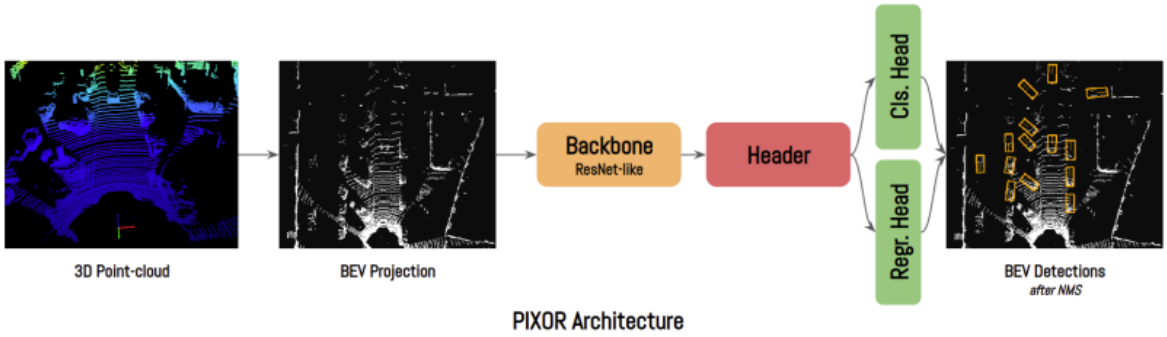


Figure 3: High-level overview for PIXOR end-to-end BEV detection framework [6]

PIXOR++ is an enhancement of PIXOR that incorporates HD maps as priors for the detector, concatenated with the input point-cloud data. These HD maps provide additional context and semantic information, improving detection accuracy, especially in challenging scenarios. PIXOR++ retains real-time performance while achieving increased detection accuracy on the KITTI benchmark.

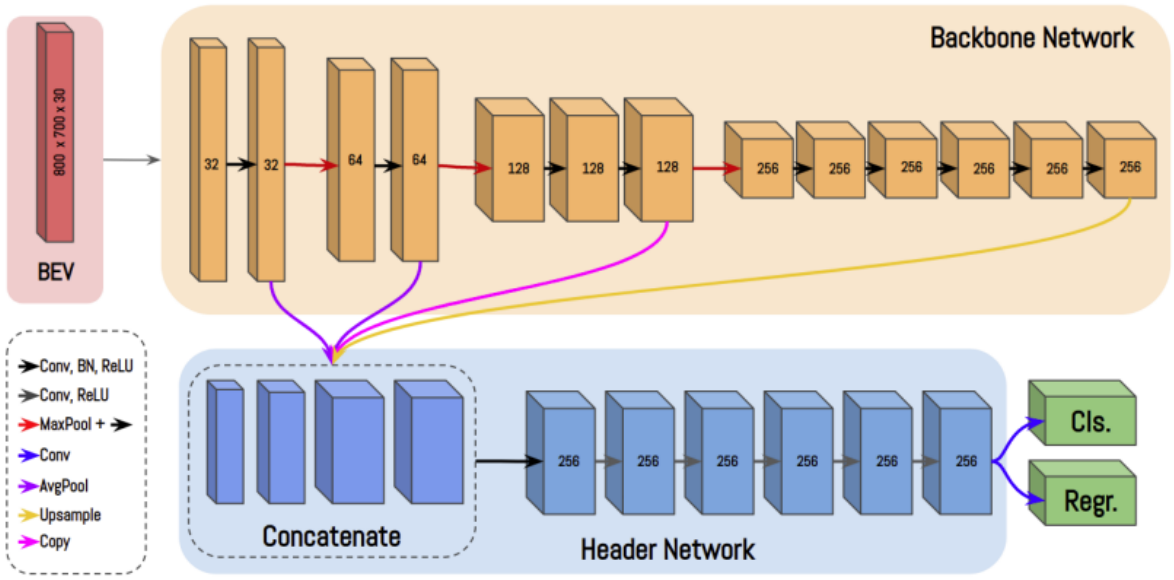


Figure 4: Architectural design for PIXOR++ end-to-end BEV detection framework (k = 3) [6]

Both PIXOR and PIXOR++ operate in the BEV domain, while another method, PointPillars, operates in the pillar-based representation. Comparison between these methods shows that PIXOR and PIXOR++ achieve competitive detection accuracy with PointPillars but at lower computational cost, making them attractive options for real-time applications.

FVNet is a 3D object detection system designed to work with raw LiDAR point clouds. Instead of directly analyzing the point cloud data, FVNet first converts it into a cylindrical frontal-view representation. This conversion simplifies the detection process by allowing the system to directly generate 3D region proposals based on the projected view.

Once the point cloud is transformed, FVNet uses a modified version of a region proposal network (RPN) to create 3D region proposals. These proposals include information about the size and position of the detected objects, as well as additional data indicating the distances to the front and back of each bounding box within the LiDAR point cloud.

After generating region proposals, FVNet crops the corresponding points from the original point cloud and processes each cropped region using a variant of PointNet. PointNet helps estimate crucial information about the bounding boxes, such as their position, size, and orientation.

To handle objects of various sizes effectively, FVNet employs a multi-scale convolutional neural network (CNN) architecture. This architecture allows FVNet to capture objects of different scales and adapt to the scale-variance issue introduced by using frontal-view maps.

2.1.4 Perception of Autonomous Vehicles

Perception is crucial for fully automated driving systems as it enables vehicles to understand their surroundings and make informed driving decisions to ensure safety and efficiency on the road. These systems rely on various sensors positioned around the vehicle to gather information about the environment. Computer vision techniques are then employed to analyze this data, detecting, localizing, and classifying objects of interest in the vehicle's vicinity.

Perception can be achieved using individual sensor modalities or by combining multiple sensors to compensate for their individual limitations. In the following sections, an overview of computer vision will be provided, and the tasks of object detection and recognition in both two- and three-dimensional spaces will be discussed.

- **Compute vision:**Computer Vision (CV) is a branch of computer science focused on enabling computers to understand and extract useful information from visual digital data, such as images. While humans perceive visual information effortlessly, implementing perception in dynamic environments is complex for computers. Computer vision researchers aim to enable computers to interpret visual data in the same way humans do.

  In the 1960s, experts believed that teaching computers to "see" would be simple. However, after sixty years, computer vision remains challenging due to several factors:

  - Limited understanding of human biological vision.
  - Complexity of the visual world, including variations in size, aspect ratio, lighting conditions, and occlusions.
  - Lack of context, as machines only have access to two-dimensional digital images.

  Despite these challenges, computer vision has transformed various industries, including security, healthcare, and automotive. Traditionally, computer vision involved designing filters and using image processing techniques to recognize patterns in images. Machine learning algorithms were then applied to process features and solve specific problems.

  In 1989, Convolutional Neural Networks (CNNs) were introduced, revolutionizing object recognition. CNNs, inspired by the brain's visual cortex, significantly improved performance in tasks like image recognition. In 2012, CNNs and deep learning were used to win the ImageNet Large Scale Visual Recognition Challenge, leading to further advancements in computer vision and deep learning.

- **Convolutional Neural Networks:**Convolutional Neural Networks (CNNs) are a type of deep neural networks designed to automate feature engineering in the context of processing visual data, such as images. Unlike traditional machine learning approaches, which rely on

manually engineered features, CNNs learn to extract features directly from input images using a gradient-based learning algorithm. This makes CNNs highly efficient and effective for tasks in computer vision.

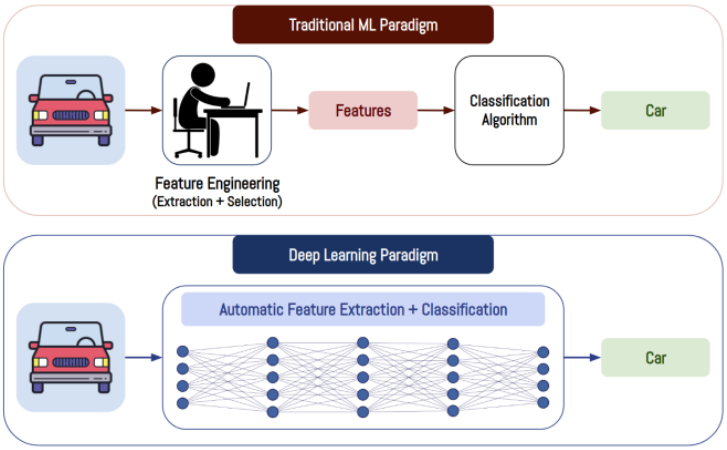


Figure 5: Difference between traditional machine and deep learning [6]

CNNs are different from other neural network architectures, such as Multi-Layer Perceptrons (MLPs), in their structure and operation. MLPs, also known as fully connected artificial neural networks, are not well-suited for processing visual data because they lack shift-invariance. This means that MLPs act differently to shifted versions of the same input, making them less effective for tasks like image recognition.

In contrast, CNNs have a shared-weights architecture, where each layer consists of a set of filters that learn to extract features from the entire input rather than focusing on specific regions. This allows CNNs to adapt to variations in input images and maintain consistent performance. Additionally, CNNs are more scalable compared to MLPs, as increasing the depth or width of the network does not significantly increase the number of learnable parameters.

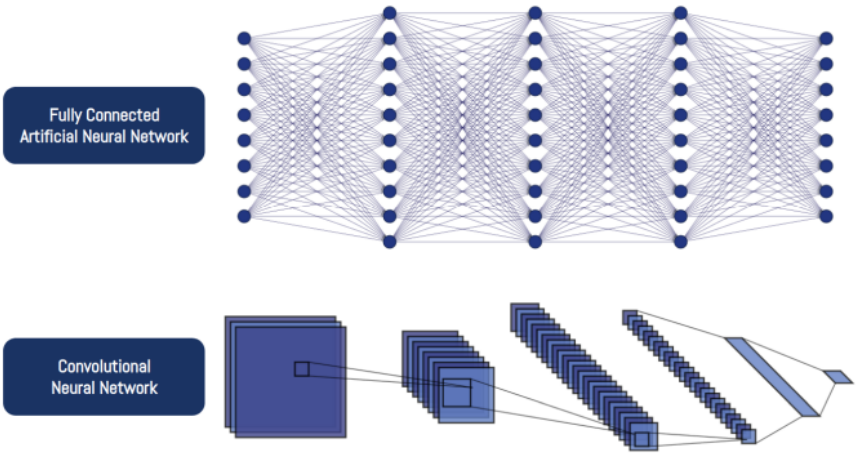


Figure 6: Architectural difference between a fully connected artificial and a convolutional neural network [6]

CNNs come in different types for processing various forms of input data, including 1D convolutions for signals, 2D convolutions for images, and 3D convolutions for volumetric data. In the field of computer vision, CNNs have led to significant advancements, particularly in tasks like object detection and recognition, which are essential components of perception in autonomous vehicles.

- **2D Object Detection and Recognition:** They are essential tasks in computer vision, especially for creating reliable perception systems. Object detection involves identifying the presence of specific objects in a scene and determining their positions, while object recognition involves labeling the detected objects and assigning them to their respective categories.

  In two-dimensional object detection and recognition tasks, Convolutional Neural Networks (CNNs) have achieved state-of-the-art performance in terms of accuracy and speed. These networks are trained to analyze RGB images and generate bounding boxes around detected objects. These bounding boxes provide information such as the object's width, length, and center in the image, along with a confidence score indicating the network's certainty about the detection and the class label of the object.

  However, detections obtained from these architectures lack depth information, which refers to the distance of the detected objects from the sensor. This depth information is crucial for understanding the spatial layout of objects in the scene.

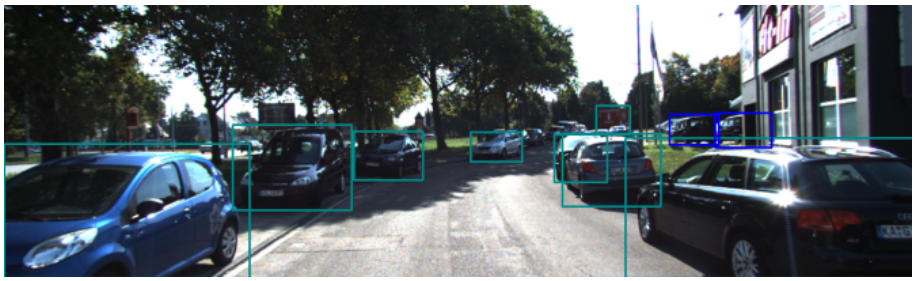


Figure 7: Example of two-dimensional vehicle detection from KITTI dataset [6]

- **3D Object Detection:** It is crucial for self-driving vehicles to understand the obstacles present in their three-dimensional environment, including vehicles, pedestrians, and cyclists. This understanding significantly improves the safety of the decision-making process. 3D object detection algorithms use Convolutional Neural Networks (CNNs) and deep learning techniques to detect and categorize objects in a scene while also predicting their distance from the vehicle, their 3D shape and size, and their orientation.

  To achieve high detection accuracy, these algorithms typically use sensor modalities that provide depth information, such as stereo cameras, RGB-D cameras, or LiDAR. This helps to assess accurately the spatial layout of objects in the environment. In the visualization provided in figure,3D bounding boxes around vehicles are depicted within a scene captured from the KITTI dataset.

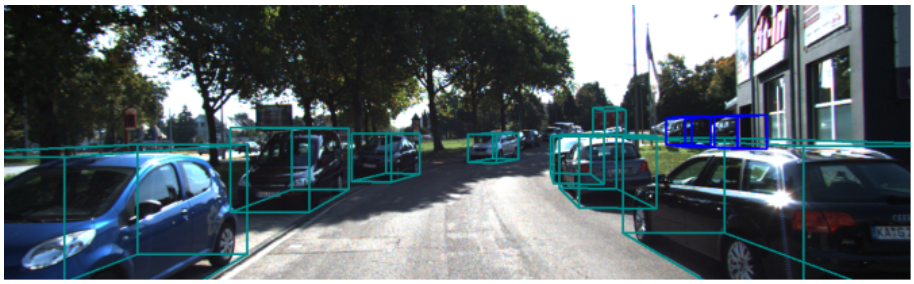


Figure 8: Example of three-dimensional vehicle detection [6]

2.2 Challenges

The main challenge lies in the amount of data required to train 3D object detection algorithms. Additionally, system reliability and processing speed are critical for autonomous driving safety. A balance between accuracy and computational cost of perception systems is required to ensure optimal real-time performance.

- **Data Volume:** Gathering and labeling data for 3D object detection is complex and time-consuming. It involves putting sensors on vehicles, calibrating them, synchronizing data, collecting the data, and validating it. Large amounts of data are needed to train 3D object detection algorithms effectively. Many companies are now providing driving data to help address this need.

- **System Reliability:** Some sensor types may fail in certain conditions, as for example cameras in low light. A reliable autonomous driving system must handle such failures and maintain accurate perception in various scenarios and environments.

- **Processing Speed and Cost:** They must make real-time decisions, requiring low-latency processing. The computational cost of the perception system is crucial, as it affects real-time performance. Choosing sensor modalities with lower computational demands is essential. Edge computing, with dedicated hardware within the vehicle, is often used to meet these requirements.

2.3 Problem definition

The study involves a detailed examination of VoxelNext's architecture and the adjustments made to ensure compatibility with the target datasets. The components of VoxelNext are analyzed, and the customization process is discussed comprehensively. Through extensive experimentation and analysis, the goal is to evaluate VoxelNext's performance and understand its challenges in real-world scenarios.

By exploring the structure and customization process of VoxelNext, functionality and effectiveness are examined thoroughly. Comprehensive tests and evaluations are conducted to assess performance accurately. The objective is to identify obstacles and strategies, providing valuable insights into the model's behavior in practical scenarios.

2.4 Contributions

VoxelNext's architecture is adapted for compatibility with datasets like nuScenes and KITTI, with an analysis of its components and customization process. Through experiments, VoxelNext's performance is assessed, and obstacles are identified, offering insights for future development in 3D object detection.

3 VoxelNeXt

3.1 Overview

In VoxelneXt, the focus shifts from handcrafted features to end-to-end training for 3D object detection. The objective is to generate 3D bounding boxes around objects and classify them based on a 3D point cloud, which consists of a set of points in 3D. The VoxelNet architecture exclusively takes the 3D points as input, eliminating the need for any mapping. Remarkably, it achieves 3D object detection by only processing the raw 3D point cloud data.

It was created for 3D object detection by turning LiDAR data into evenly spaced 3D voxels, where each voxel represents a group of points in 3D space. Each voxel is then encoded using the voxel feature encoding layers and then pass to a group of convolutional layers prior to or region proposal network, which predicts our 3D bounding box.

The core idea behind VoxelNeXt is to predict box features directly from sparse voxel features. This key advantage eliminates the need for anchor proxies, sparse-to-dense conversion, region proposal networks, and other complicated components.

Moreover, VoxelNeXt is effective in distinguishing between multiple object classes during training by considering their dimensions and proportions. This includes analyzing the ratios of length, width, and height, which helps accurately categorize objects even when they have similar shapes. Additionally, VoxelNeXt incorporates distance information, categorizing objects based on their proximity to the sensor, whether they are near, mid-range, or far. This comprehensive approach improves the model's ability to identify objects of different classes and predict their bounding boxes accurately in various spatial contexts.

Additionally, VoxelNeXt is very efficient in navigating complex environments without the need for complicated conversions or postprocessing steps. This efficiency, along with its precise object detection capabilities, makes VoxelNeXt a top choice in autonomous driving technology.

Below is the general outline of the program compared to mainstream 3D detectors. Mainstream 3D detectors use dense features and manually marked anchor detectors, as previously discussed.

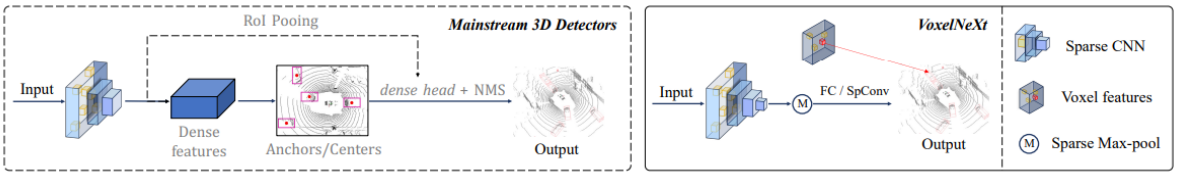


Figure 9: Comparative between mainstream 3D detectors and VoxelNeXt [1]

3.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) play a pivotal role in advancing the perception systems of autonomous vehicles. Traditionally, computer vision relied on manual feature engineering and conventional machine learning techniques. However, as the field progressed, the complexity of real-world scenarios demanded more adaptive solutions.

Deep learning revolutionized computer vision, with CNNs being the base of modern perception systems. Unlike traditional methods, CNNs autonomously learn features from data, enabling them to recognize diverse patterns effectively. The remarkable success of CNNs led to significant advancements in computer vision and deep learning.

CNNs have become indispensable in improving the perception capabilities of autonomous vehicles, driving substantial progress in sensor fusion and object recognition.

3.2.1 Basic Building Blocks

- **Convolutional Layer**

The convolutional layer is a key part of CNNs, essential for extracting features from data. It is made up of learnable filters, called kernels, which perform convolution operations. These filters are like small images designed to spot specific features in the input, like edges.

During a convolution operation, a filter slides over the input image, multiplying values at each position to create a feature map. This process detects features by matching filters with local structures, resulting in varying values based on feature presence. CNNs use many filters to extract different features from input data.

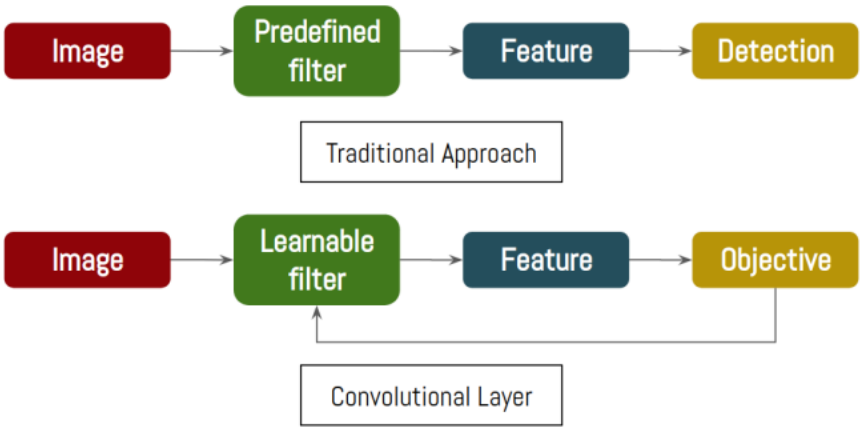


Figure 10: Tradition approach for feature extraction vs the mode of operation of a convolutional layer within a deep neural network [6]

Several important parameters control the convolutional layer's output:

- Filter size (K): Determines filter dimensions, often odd numbers for centering.
- Number of filters (F): Dictates how many patterns a layer can extract.
- Stride (S): Sets step size for filter movement, affecting output size.
- Padding (P): Adds extra pixels to maintain input dimensions, often zero-padding.

Output dimensions are calculated using input size, filter parameters, stride, and padding. Each convolutional layer also has a bias term, adding to the total learnable parameters. These parameters can be calculated using the formula: $F \times K \times K \times C + F$, where F is filters, K is kernel size, and C is input channels, as it can be seen in the next image:

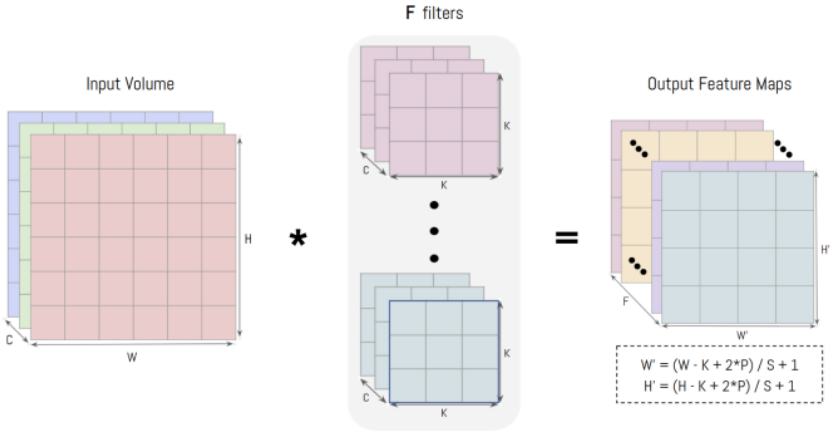


Figure 11: Demonstration of a convolution operation [6]

- **Non-linearity Operation**

In convolutional neural networks (CNNs), the non-linearity operation introduces complexity, enabling networks to model intricate data relationships. Initially, Sigmoid and Tanh activations were common but faced challenges like slow convergence. ReLU emerged as a more efficient alternative due to its simplicity, faster computation, and ability to prevent saturation. By filtering out irrelevant information, ReLU empowers CNNs to focus on essential features, crucial for tasks like image classification and object detection.

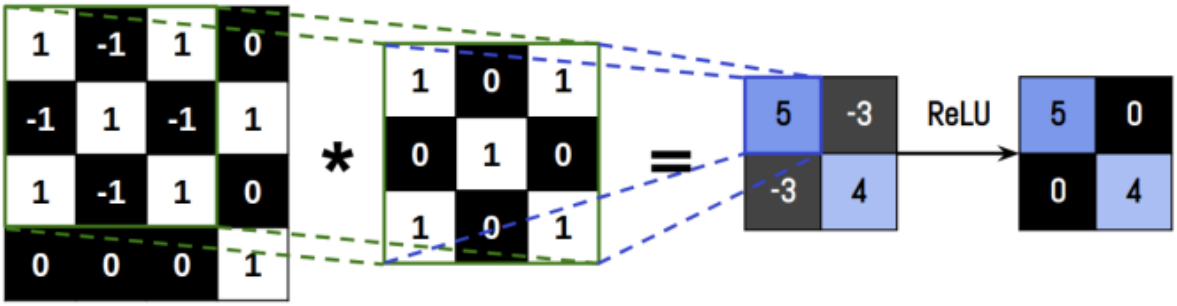


Figure 12: Convolution operation followed by a ReLU activation function [6]

- **Downsampling Operation**

It reduces input dimensions, crucial for preserving relevant information and reducing computational complexity. It is achieved through pooling layers or strided convolutions.

Pooling layers involve sliding a window over input data, applying operations like average or maximum to produce single-valued outputs. Max pooling is preferred for downsampling due to its better performance and computational efficiency. Pooling layers have no learnable parameters, making them computationally inexpensive.

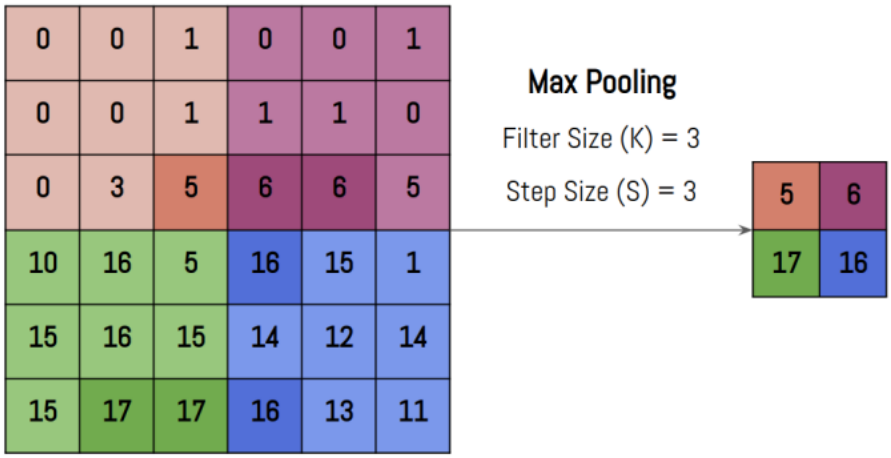


Figure 13: Example of max pooling operation [6]

3.2.2 CNN Architecture Design

The design of convolutional neural network (CNN) architecture involves critical decisions in several key areas:

- **Objective Function:**

The choice of objective function depends largely on the available data and the type of task being addressed. In supervised learning the objective function seeks to minimize the error between the model's predicted outputs and the expected outputs from the training data. This is achieved by optimizing a set of learnable parameters based on the training data. Supervised learning tasks are commonly categorized into classification and regression, each with its own specific activation functions and loss functions.

- **Optimization**

It involves finding the values of the model's parameters that minimize the loss function. Gradient descent algorithms iteratively update the model's weights in the direction that reduces the loss. Choosing the batch size and learning rate is crucial for stable training and effective convergence.

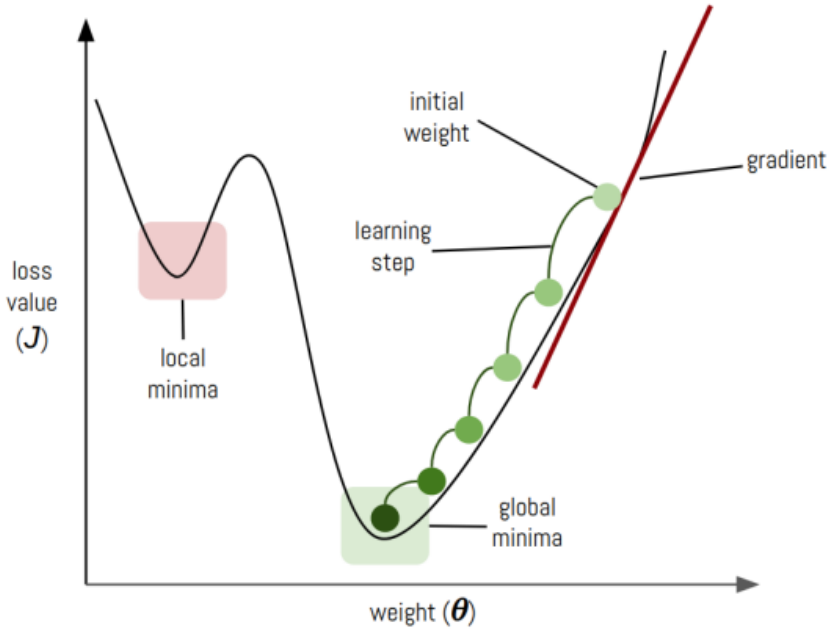


Figure 14: Gradient-based optimization algorithm [6]

- **Regularization**

To prevent overfitting in large neural networks, regularization techniques are applied. These include L1 and L2 norm penalties, data augmentation, and dropout. Regularization helps improve the model's generalization and performance on unseen data.

- **Normalization**

They address the issue of internal covariant shift in deep neural networks. They stabilize training and improve convergence speed by providing a more stable data distribution at each layer.

- **Width and Depth**

Depth refers to the number of CNN blocks, while width refers to the number of filters in each block. Deeper networks can learn more complex functions but may face challenges such as vanishing gradients. The width of the network varies at each layer to effectively extract relevant features at different levels of abstraction.

3.2.3 CNN Backbone Adaptation

In the realm of 3D object detection, VoxelNeXt and CNN backbone adaptation work together to solve the challenges of understanding complex 3D scenes. Below is explained how they help each other.

VoxelNeXt Integration with CNN Backbone:
VoxelNeXt relies on the backbone architecture of a convolutional neural network to process and analyze voxelized point cloud data. The CNN backbone provides the foundational framework for feature extraction and subsequent detection tasks.

Through CNN backbone adaptation, the architecture of the convolutional neural network is tailored to accommodate the unique characteristics of 3D object detection tasks, such as processing sparse and irregular point cloud data.

Feature Extraction and Encoding:
The CNN backbone, adapted for 3D object detection, plays a crucial role in feature extraction from the input point cloud data. It processes voxelized features, capturing essential geometric and spatial information.

VoxelNeXt then further refines these features through specialized encoding layers, such as PointNet, to extract higher-level representations of the input data, facilitating accurate object detection.

Efficient Sparse Convolution:
VoxelNeXt leverages sparse convolutional operations to efficiently process the encoded voxel features. These sparse convolutions operate exclusively on essential locations within the voxel grid, optimizing computational resources.

By integrating sparse convolutional operations with the adapted CNN backbone, VoxelNeXt achieves a balance between computational efficiency and feature complexity, essential for real-time 3D object detection tasks.

Enhanced Object Detection and Tracking:
The fusion of VoxelNeXt and CNN backbone adaptation enables robust object detection and tracking in complex 3D environments. The adapted CNN backbone provides a solid foundation for feature extraction, while VoxelNeXt enhances detection accuracy through direct prediction of 3D objects from voxel features.

This collaborative approach ensures precise localization and classification of objects, critical for autonomous driving and other applications requiring accurate 3D perception.

Training and Fine-Tuning:
During training, the adapted CNN backbone is fine-tuned using annotated training data, allowing the network to learn to recognize and localize objects accurately.
Fine-tuning techniques, such as transfer learning and domain adaptation, further refine the performance of the adapted CNN backbone on specific 3D object detection tasks and datasets, ensuring optimal results in real-world scenarios.

The integration of VoxelNeXt with CNN backbone adaptation represents a good approach to 3D object detection, combining the strengths of both methodologies to achieve accurate and efficient detection in complex 3D environments.

3.3 Detailed designs

The following chapters will delve deeper into how VoxelNeXt operates.

Four detailed designs contribute to performance and efficiency, that can be seen in the following image.

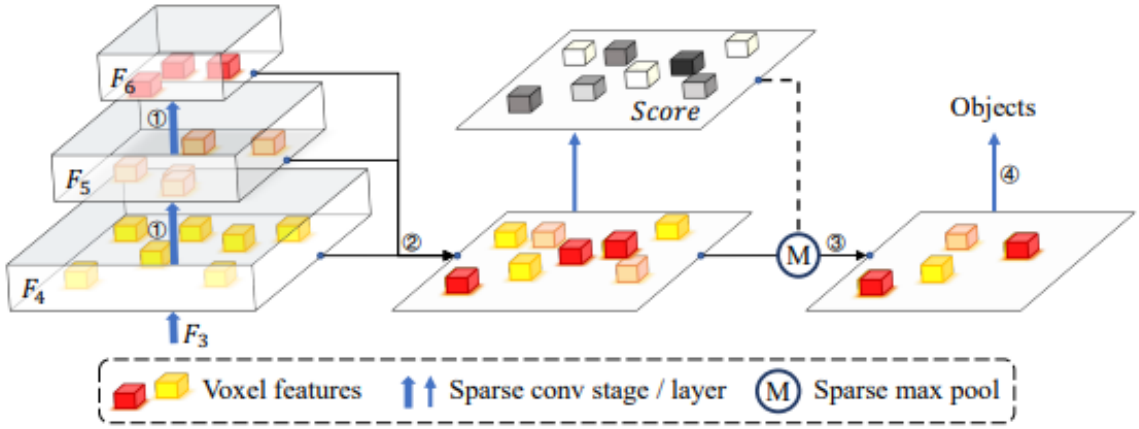


Figure 15: Detailed designs [1]

3.3.1 Detail 1: Two additional down-samplings

The first improvement involves adding two extra layers that make the sparse backbone deeper. This is important because having strong features with big coverage areas is necessary for making accurate predictions directly from sparse voxel features. Some recent techniques suggest different ways to improve the sparse backbone using dynamic convolution, big filters, and transformers. However, it is chosen to keep things simple and only add a few extra layers. When it was visualized, the effect of these added layers, we see that the coverage areas become larger, resulting in more accurate predictions for the boxes. This improvement is effective and doesn't require much extra computation.

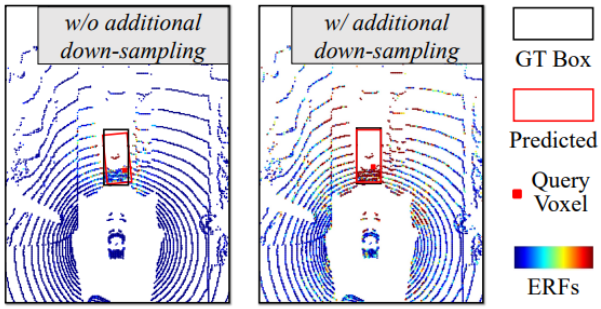


Figure 16: Effects of additional down-sampling layers on effective receptive fields (ERFs) and the predicted boxes [1]

3.3.2 Detail 2: Sparse Height Compression

In VoxelNet, it was found that 2D sparse features are efficient for prediction. In VoxelNeXt, height compression is fully sparse. Voxels are simply placed on the ground and features in the same positions are summed up. It was observed that making predictions based on these compressed 2D sparse features costs less than using 3D ones.

Method	Backbone	Head	Latency	mAP	NDS
-	3D	3D	92 ms	56.3	63.4
VoxelNeXt	3D	2D	66 ms	**56.2**	**64.3**
VoxelNeXt-2D	2D	2D	**61 ms**	53.4	62.6

Figure 17: Ablations on 2D or 3D sparse CNN in VoxelNeXt. sparse height Compression is used to connect 3D backbone and 2D head [1]

3.3.3 Detail 3: Spatially Voxel Pruning

In 3D scenes, a large number of redundant background points that contribute minimally to predictions are often present. To address this, unnecessary voxels are gradually removed as layers are down-sampled. This voxel pruning results in a significant reduction in computation without impacting performance.

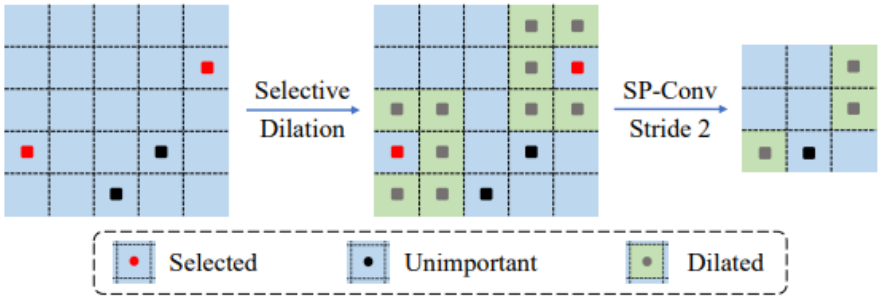


Figure 18: Spatially voxel pruning [1]

3.3.4 Detail 4: Sparse Max Pooling

During inference, NMS post-processing can be avoided by employing sparse max pooling, given the sparse nature of the features. Sparse max pooling is utilized to select voxels with localized maximum values. The exclusion of removed voxels from box prediction results in computational savings for the head.

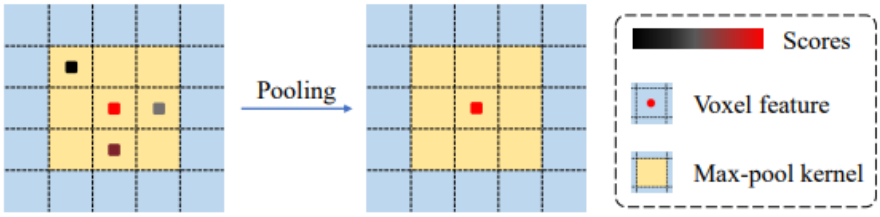


Figure 19: Sparse max pooling layer [1]

3.4 Better tracking

The framework naturally extends to 3D tracking. The position of the voxel used to predict each box is recorded. Similar to center association, the L2 distance for matching is computed. Empirical evidence shows that voxel association improves tracking.

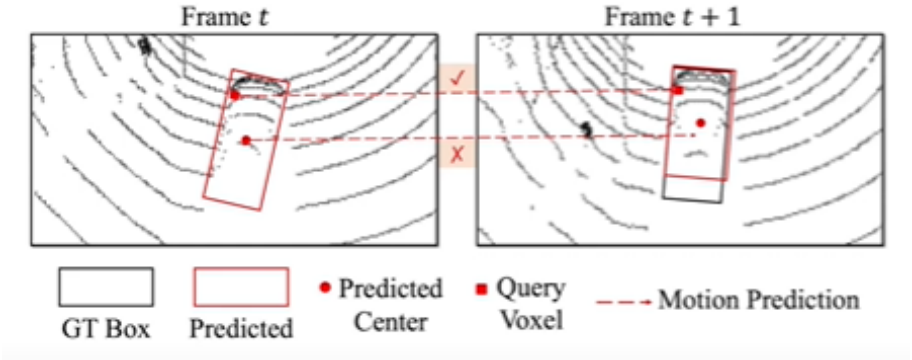


Figure 20: Visualization of voxel association [1]

3.5 Not required centers

In VoxelNeXt, voxels used for box prediction aren't required to be located inside the boxes or their centers. Instead, the relative positions of voxels within the 3D bounding box they generate are counted. Interestingly, statistically, only a small percentage of boxes (less than 10 percent in total) are predicted based on voxels near object centers.

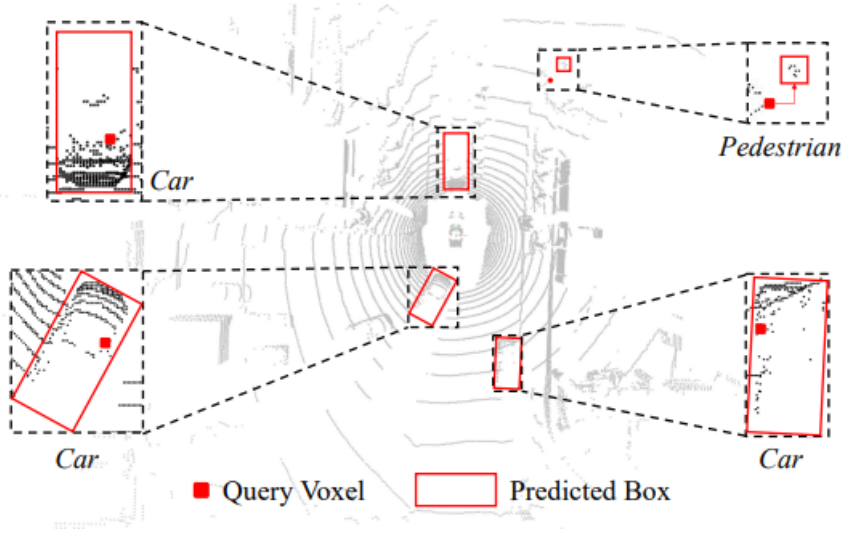


Figure 21: Visualization on the predicted boxes and their query voxels [1]

3.6 VoxelNeXt better that Center Point

The CenterPoint framework presents a streamlined methodology for 3D object detection, designed to efficiently process LiDAR point-cloud data. Initially, a standard 3D backbone is employed to extract feature representations from the input data, facilitating subsequent analysis.

Following this, a 2D Convolutional Neural Network (CNN) architecture serves as the detection head, tasked with identifying object centers within the feature representations and generating initial 3D bounding box estimates based on these center features. This step essentially involves pinpointing object locations and approximating their 3D shapes and sizes.

Subsequently, point features are extracted from the 3D centers of each face of the estimated bounding boxes. These features undergo further refinement through a Multi-Layer Perception (MLP), which fine-tunes the box regression and predicts confidence scores using Intersection over Union (IoU) as a guiding metric.

The framework overview diagram visually represents this process.

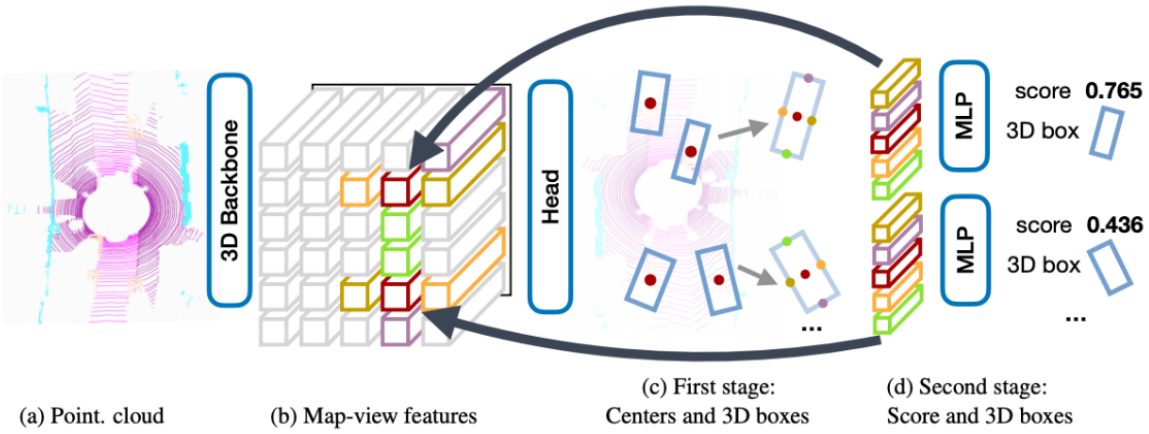


Figure 22: CenterPoint framework [4]

VoxelNeXt demonstrates a notable 4.9 percentage decrease in orientation error compared to CenterPoint, possibly due to the enhanced sensitivity of sparse voxel features to orientation disparities. It achieves superior detection performance, surpassing existing LIDAR methods on the nuScenes tracking test benchmark and delivering commendable results on the Waymo dataset. Argoverse2 is a long-range dataset with a perception range of 200m. The dense-head method, CenterPoint, experiences high latency because of its large dense BEV feature map. In contrast, the fully sparse method demonstrates significantly better performance.

Moreover, our work with the KITTI Dataset involved training and evaluating the model for 40 epochs with three classes (Cars, Pedestrians, and Cyclists), resulting in few false detection observed further away from the LIDAR sensor. Fine-tuning efforts were undertaken to rectify these false positives, and the updated model's sample images and predicted results are provided for reference. The results will be discussed in greater depth in the next section.

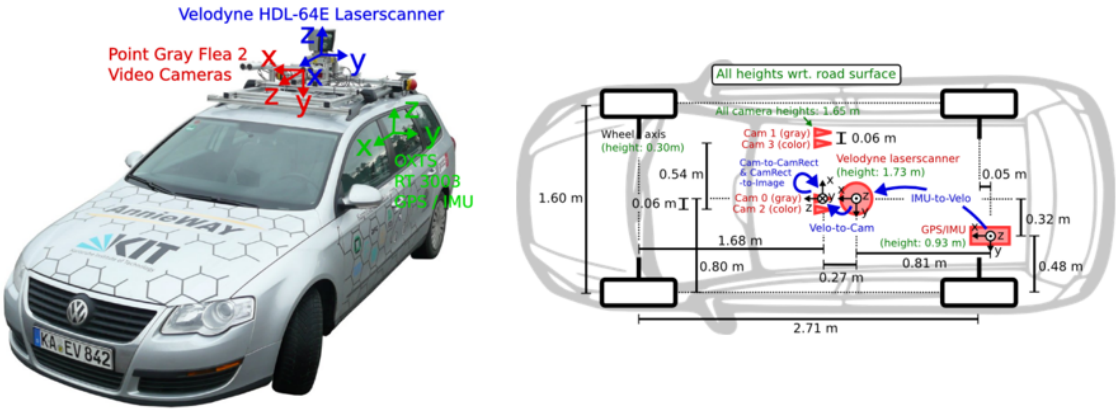
4 Implementation

4.1 KITTI Dataset

4.1.1 KITTI Dataset Overview

The KITTI Vision Benchmark Suite offers an excellent set of datasets for automotive applications such as car, cyclists and pedestrian recognition.These datasets include optical flow, stereo vision, visual odometry, SLAM, and object detection. KITTI object detection dataset is one of the most common dataset for driving scenes collected in the daytime and under favorable weather conditions. This dataset contains 7,481 training samples and 7,518 testing samples for both images (with a resolution of 1242 × 375) and point cloud. The training dataset is divided into a training part (3,712 samples) and a validation part (3,769 samples).

As shown in next figure, the camera and the LiDAR utilize two distinct coordinate systems (red for the camera and blue for the LiDAR). The directions (X, Y, Z) are set as (rightward, downward, forward) and (forward, leftward, upward) from the camera and LiDAR, respectively.



(a) Equipped Vehicle Volkswagen Passat B6 (b) Bird's eye view (BEV)

Figure 23: KITTI vehicle Setup [13]

An example from KITTI dataset in shown in Figure below.

Figure 24: KITTI Image Example (RGB: 1224 x 37 px)

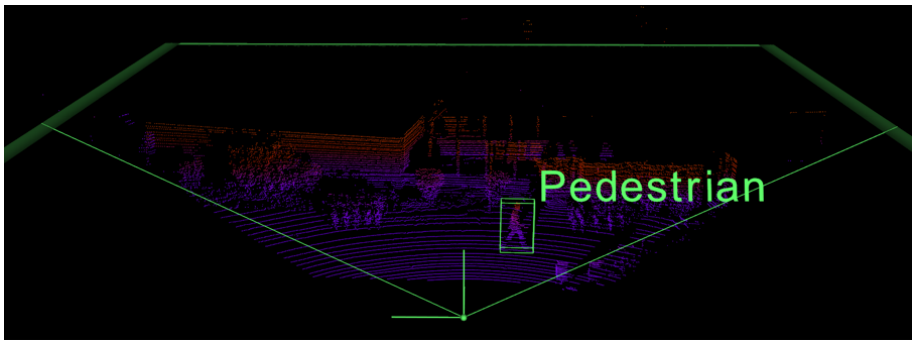


Figure 25: KITTI Point Cloud corresponding to the RGB image (Filtered)

Points and objects that are not in range of the front camera are filtered out following the BEV range (side range of [-40 m, 40 m] and forward range of [0 m, 70.4 m]), as shown in Figure 25.We only focus on objects that appear in the image (left camera). As a result, it is necessary to perform filtering on the point cloud to keep those in the camera's field of view.

KITTI divides the world into eight categories:car, van, truck, pedestrian, person sitting, cyclist, tram, and misc. Our training concentrate only on the 3 classes Car, cyclists and pedestrians data category.

The difficulty of detecting objects varies depending on the situation. Some items, such as occluded objects, long-range objects, and so on, are more difficult to detect. The farther away an object is from the vehicle, the smaller it appears in the RGB image and the fewer points it reflects back into the point cloud. As shown in next table, the KITTI object detection benchmark has three difficulty levels to evaluate the object detection system. The object size in the image, occlusion, and truncation distinguish these three levels (Easy, Moderate and Hard).

	Easy	Moderate	Hard
min. 2D box height	40 pixels	25 pixels	25 pixels
max. occlusion level	fully visible	partly occluded	difficult to see
max. truncation percentage	15	30	50

Table 1: KITTI 3D object detection benchmark's object attribute parameters for the three difficulty levels.

4.1.2 Training Configuration for KITTI Dataset

The configuration file for the KITTI dataset contains critical parameters essential for training an object detection model. It comprises sections dedicated to data configuration, augmentation techniques, model architecture, target assignment, loss calculation, post-processing, and optimization strategies. The different sections are explained in details below:

1. **Data Configuration**
   This section lists the different categories or classes present in the dataset. In this case, the KITTI dataset includes three main categories: 'Car', 'Pedestrian', and 'Cyclist'. These classes represent the types of objects that the model will be trained to detect.
   Here, the data configuration files that contains additional dataset specific settings are defined. The major parameters used during dataset preparation are listed below:

   - Training to validation dataset split: 1:1
   - Point cloud range: X-axis [0,70], Y-axis [-40,40], Z-axis [-3,-1]
   - Voxel size: [0.05, 0.05, 0.1]
   - Maximum points per voxel: 5

2. **Data Augmentation**
   This section defines various data augmentation techniques applied during training to increase dataset variability and enhance model robustness. Augmentation techniques include ground truth sampling, which adjusts the dataset distribution to alleviate class imbalance, and transformations like flips, rotations, and scaling are applied to both input data and annotations.

3. **Model Configuration**
   Here we specify the architecture and components of the object detection model. The VFE (Voxel Feature Encoding) defines the method used to encode voxel features. The backbone_3d specifies the backbone architecture, which is responsible for extracting hierarchical features from the input data and Dense_head configures the head of the model, responsible for generating predictions for each object. Parameters like the input features, class-specific settings, and convolutional configurations are defined here. In this case, we used the architecture defined by the paper. i.e VoxelResBackBone8xVoxelNeXt and VoxelNeXtHead.

4. **Target Assignment and Loss Calculation**
   In this section, we define how ground truth annotations are assigned to predicted object proposals during training. Parameters like feature map stride, maximum objects per sample, and Gaussian overlap threshold are specified here. It also includes settings such as loss weights for classification and localization, which determine the relative importance of these tasks during training.

5. **Post-processing**

   This part deals with the steps to be performed after model inference. This includes filtering out predictions below a certain score threshold, restricting object positions within specific bounds, and applying non-maximum suppression to remove redundant detection. The Non Maximum Suppression (NMS) was kept at 0.5 and Recall were measured for three Intersection of Union (IOU). i.e 0.3, 0.5 and 0.7.

6. **Hyper-parameters and optimization:**

   We define the batch size as 4 samples per iteration and number of training epochs as 40. The adam one cycle optmizer was used here. The major parameters under this field are explaind below:

   - Learning rate: It determines the model's parameters update during traning. It was kept at 0.01.
   - Weight_decay: It regulates the magnitude of parameter updates to prevent overfitting of the model. It was specified as 0.01.
   - Momentum: It controls the optimization algorithm, influencing the direction and speed of parameter updates. It was kept at 0.9
   - Learning rate scheduling: It dictates how dynamically the learning rate are adjusted during training.

4.2 NuScenes Dataset

4.2.1 NuScenes Dataset Overview

The nuScenes dataset is a large-scale autonomous driving dataset. The dataset has 3D bounding boxes for 1000 scenes collected in Boston and Singapore. Each scene is 20 seconds long and annotated at 2Hz. This results in a total of 28130 samples for training, 6019 samples for validation and 6008 samples for testing. The nuScenes dataset includes 1.4 million camera images, 400,000 LIDAR sweeps, 1.3 million radar sweeps and 1.1 million object bounding boxes in 40k key frames. The dataset has the full autonomous vehicle data suite: 32-beam LiDAR, 6 cameras and radars with complete 360° coverage. The 3D object detection challenge evaluates the performance on 10 classes: cars, trucks, buses, trailers, construction vehicles, pedestrians, motorcycles, bicycles, traffic cones and barriers. It serves as a benchmark for developing and testing algorithms in perception, localization, mapping, planning, and other critical aspects of autonomous vehicle technology.

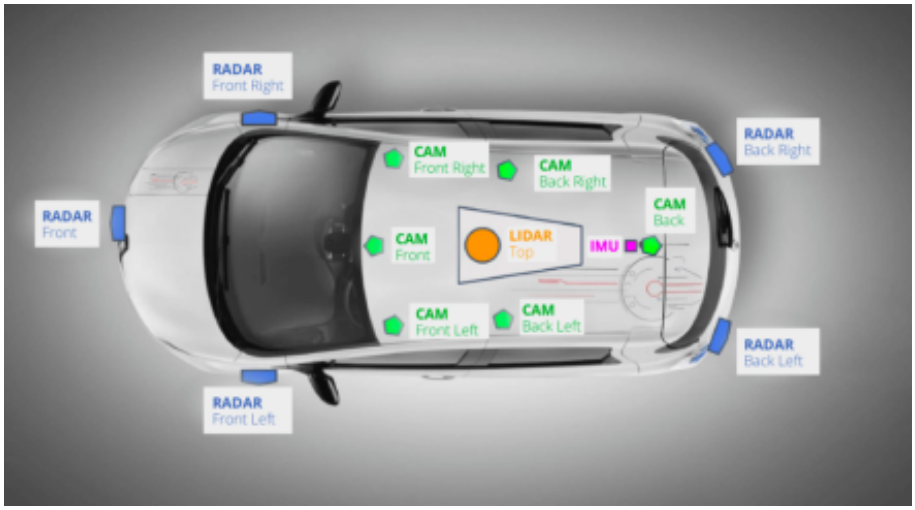


Figure 26: NuScenes Car Sensors setup

A sample of a image and ground truth 3D bounding boxes is shown below:

Figure 27: Sample Image NuScenes Dataset

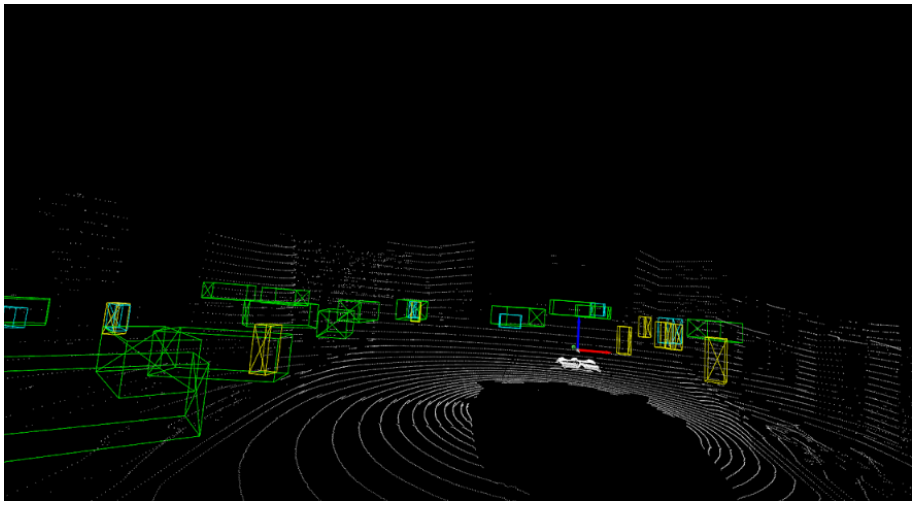


Figure 28: NuScenes LIDAR with ground truth 3D Bounding Box

4.2.2 Training Configuration for NuScenes Dataset

1. **Data Configuration:** Due to the huge size of NuScenes dataset, it is distributed into mini, trainval and test splits. Mini (10 scenes) is a subset of trainval used to explore the data without having to download the entire dataset. Trainval (700+150 scenes) is packaged into 10 different archives that each contain 85 scenes. The major parameters used during dataset preparation are listed below:

    - Training to validation dataset split: 7:3 (mini-version)
    - Point cloud range: X-axis [-54m, 54m], Y-axis [-54m, 54m], Z-axis [-5m, 3m]
    - Voxel size: [0.075m, 0.075m, 0.2m]
    - Maximum points per voxel: 10
    - Maximum sweeps: 10

2. **Data Augmentation:** For NuScenes, GT sampling and translation augmentations are used. Flipping is randomly conducted along X and Y axes. Rotation angle is randomly picked between -45 and 45 degrees. Global scaling is conducted by a factor sampled between 0.9 and 1.1. The translation noise factors are sampled between 0 and 0.5.

3. **Model Configuration:** The VFE (Voxel Feature Encoding) method is used to encode voxel features. The backbone_3d specifies the backbone architecture, which is responsible for extracting hierarchical features from the input data and Dense_head configures the head of the model, responsible for generating predictions for each object. Parameters like the input features, class-specific settings, and convolutions configurations are defined here. In this case, we used the architecture defined by the paper. i.e VoxelResBackBone8xVoxelNeXt and VoxelNeXtHead.

4. **Target Assignment and Loss Configuration:** We define how ground truth annotations are assigned to predicted object proposals during training. Feature map stride was set to 8, maximum objects per sample set to 500, and Gaussian overlap threshold was kept at 0.1. It also includes settings such as loss weights for classification and localization.

5. **Hyper-parameters and optimization:** We define the batch size as 2 samples per iteration and number of training epochs as 20. The Adam one cycle optimizer is utilized here. The major parameters under this field are explained below:

    - Learning rate: It determines the model's parameters update during training. It was kept at 0.001.
    - Weight_decay: It regulates the magnitude of parameter updates to prevent over fitting of the model. It was specified as 0.01.
    - Momentum: It controls the optimization algorithm, influencing the direction and speed of parameter updates. It was kept at 0.9
    - Learning rate scheduling: It dictates how dynamically the learning rate are adjusted during training.Learning rate decay was set to 0.1.

5 Results

5.1 NuScenes

5.1.1 Evaluation Metrics Overview

In the NuScenes dataset, there are a set of parameters or metrics used to evaluate the performance of algorythms. These parameters provide comprehensive insights into the performance of object detection and tracking algorithms, considering various aspects such as localization accuracy, scale estimation, orientation prediction, velocity estimation and attribute recognition. Evaluating algorithms based on these metrics helps to understand the strengths and weaknesses of the algorithms and guide for improvements in autonomous driving systems. The most important parameters are:

- **Mean Average Precission (mAP):** This metric is used to evaluate the accuracy of object detection algorythms. It computes the average precision (AP) for each class and then takes the mean over all classes. The AP measures the precision-recall curve for detecting objects at various thresholds and computes the area under this curve. Higher mAP indicates better performance. Its value oscillates between 0 and 1. As a result, a mAP closer to 0 will mean that the object detection is low efficient; whereas a mAP value closer to one means that the algorithm detects the objects very precisely and with no false positives.
- **NuScenes Detection Score (NDS):** NDS is a metric specifically designed for evaluating object detection algorythms on the NuScenes dataset. It considers the localization and orientation accuracy of detected objects. NDS penalizes false positives and false negatives based on their position, size and orientation with respect to ground truth annotations. The interval in which the NDS value moves is similar to the one of mAP, where a value closer to 0 means a bad detection and a value closer to 1 means a good detection.

5.1.2 Obtained Metrics

The results obtained here after training and testing correspond to the mini-version of the NuScenes dataset. Firstly, for the training 10 different classes of objects were selected at first: "Car", "Truck", "Construction Vehicle", "Bus", "Trailer", "Barrier", "Motorcycle", "Bicycle", "Pedestrian" and "Traffic Cone". Then, the same procedure was applied to only 3 classes: "Car", "Bicycle" and "Pedestrian". In both cases, the model was trained and evaluated first up to 10 epochs in first phase and increased up to 20 epochs in the next phase. The results shown below are for validation set of the mini-version of nuScenes.

Model	Epochs	mAP	NDS
VoxelNext[1]	20	60.0%	67.1%
10 Classes	10	20.2%	28.9%
	20	21.0%	29.5%
3 Classes	10	12.5%	15.9%
	20	12.8%	14.8%

Table 2: Comparision Table for mAP and NDS metric on mini-version

As some of the objects classes need more data during the training, these objects are not detected. Hence resulting in a poor mAP and NDS Score, compared with the full dataset evaluation results from the reference paper.

Model	Epochs	Car	Truck	Bus	Trailer	CV	Ped	Mot	Byc	Cone	Bar
VoxelNext[1]	20	85.6%	58.4%	71.6%	38.6%	17.9%	85.4%	59.7%	43.4%	70.8%	68.1%
10 Classes	10	64.9%	34.9%	16.8%	0.0%	0.0%	77.6%	7.5%	0,0%	0,0%	0,0%
	20	66.5%	31.0%	26.2%	0.0%	0.0%	77.9%	8.6%	0,0%	0,0%	0,0%
3 Classes	10	51.1%	-	-	-	-	73.7%	-	0.0%	-	-
	20	56.9%	-	-	-	-	71.1%	-	0.0%	-	-

Table 3: Comparision Table for mAP metric for each classes on mini-version

The results for individual classes is shown below, where we can see the accuracy scores of 5 classes "Trailer", "Construction Vehicle", "Bicycle", "Pedestrian" and "Traffic Cone" are null as these cannot be predicted due to the limited data.

However, the results for classes "Car" and "Pedestrian" were satisfactory. The training was done in two phases, firstly trained and evaluated for 10 epochs and the same for the next 10 epochs.

We attempted to adjust the training setup specifically for three classes: "Car," "Pedestrian," and "Bicycle." However, we observed a decline in accuracy specifically for the "Car" class. This drop in accuracy may be attributed to an increase in false positives, where the model confused "Truck" and "Bus" instances as "Car."

5.2 KITTI Dataset

5.2.1 Evaluation Metrics Overview

Here, we explain the different evaluation terms such as the Intersection-over-Union (IoU), precision rate (Precision), recall rate (Recall), and average accuracy (AP). The IoU measures the degree of coincidence between the network prediction target box and the original real target box, which can be expressed as.

$$IOU = \frac{\text{Detection result} \cap \text{Ground Truth}}{\text{Detection result} \cup \text{Ground Truth}}$$

In the numerator we compute the area of overlap between the predicted bounding box and the ground-truth bounding box. The denominator is the area of union, or more simply, the area encompassed by both the predicted bounding box and the ground-truth bounding box.

The role of the IoU is to help resolve the correctness of the detection results, and the general algorithm sets a threshold of T that is considered correct when the IoU is greater than T is otherwise judged incorrect. On this basis, the prediction results can be divided into the following four categories:

- TP (True positives): Positive samples were identified as positive samples.
- TN (True negatives): Negative samples were identified as negative samples.
- FP (False positives): Negative samples were misidentified as positive samples.
- FN (False negatives): Positive samples were misidentified as negative samples.

On the basis of the IoU, the definitions of Precision and Recall can be given. Precision is actually among all the predicted targets, predicting the correct ratio whereas Recall the proportion of samples correctly identified as positive samples out of all positive samples in the test set.

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

AP, which can be calculated using Precision and Recall, is actually the area under the Precision–Recall curve. Generally, the better the target detection model is, the higher its AP value will be. To contrast this with existing algorithms, AP is also used as the main detection index.

5.2.2 Obtained Metrics

The results for KITTI were evaluated on validation set, and the compared with other state of art models that use only point clouds for 3D object Detection. Since, most models use the Average Precision (AP)@ Recall over 40 positions at 0.7 IOU thresholds to compare and analyze the results, we included the datas for the same for the comparision in below table.

Comparison of 3D Detector Results Average Precision @ Recall 40 positions Validation set										
Model	Reference	Car 3D			Pedestrians			Cyclists 3D		
VoxelNet [3]	CVPR 2018	81.97	65.46	62.85	57.86	53.42	48.87	67.17	47.65	45.11
Point Pillars [14]	IEEE 2019	84.09	75.13	69.43	62.57	57.52	51.17	81.96	62.11	57.39
PV-RCNN [16]	CVPR 2020	92.57	84.83	82.69	64.26	56.67	51.91	88.88	71.95	66.78
SE-SSD [17]	CVPR 2021	93.19	86.12	83.31	67.98	59.72	54.83	91.77	72.54	68.78
GLENet [18]	IJCV 2023	93.51	86.1	83.6	69.55	64.12	59.23	92.77	72.44	68.11
BtcDET [19]	AAAI 2022	93.15	86.28	83.86	69.39	61.19	55.86	91.45	74.7	70.08
TSSTDet [14]	IEEE 2024	95.29	89.06	86.92	75.13	69.38	64.31	95.16	76.24	71.62
VoxelNeXt	Ours	87.81	78.89	76.42	62.15	57.03	53.78	87.34	67.13	64.15

Table 4: KITTI Validation set results comparison with other models

The above results for VoxelNext model is computed for 40 epochs. The training loss was steady so we proceeded to the evaluation phase. The resulting metrics for validation was computed and visualization was done on random sample point cloud data of the testing set. The results were found it to be accurate for the objects near the vicinity of the sensor whereas, few false detection were seen far away from the LIDAR sensor.

Comparing the performance with other state of art models the recent development in the field of LIDAR-only object detections, VoxelNext model trained under our defined configuration stands in the average range of 3D detection model zoo. Continuing to train for further epochs by implying further augmentation techniques (for e.g double flip) could possibly yield better results.

We generated the predictions for the KITTI test set and sent an application for the results metrics on official KITTI Evaluation Benchmark site. We are yet to receive the response, in order to incorporate and compare the results with other state of art models. We have uploaded the prediction results file onto our GitHub repository.

Here, we list other evaluation metrics such as Average Precision (AP)@ Recall over 11 positions, Bird Eye View (BEV) Detection and Average Orientation Estimation (AOS) for the moderate category.

3D Detector Results @ IOU 0.7 Validation set			
Metrics	Car 3D	Pedestrian 3D	Cyclists 3D
AP @ R11	77.22	56.86	67.32
BEV	87	63.42	69.54
AOS	89.33	72.77	77.06

Table 5: KITTI Validation set results AP, BEV and AOS

5.2.3 Visualization: KITTI Testing Set

Green Bounding Box: Cars
Yellow Bounding Box: Cyclists
Cyan Bounding Box: Pedestrians

Figure 29: Sample Image Test Dataset(000744)

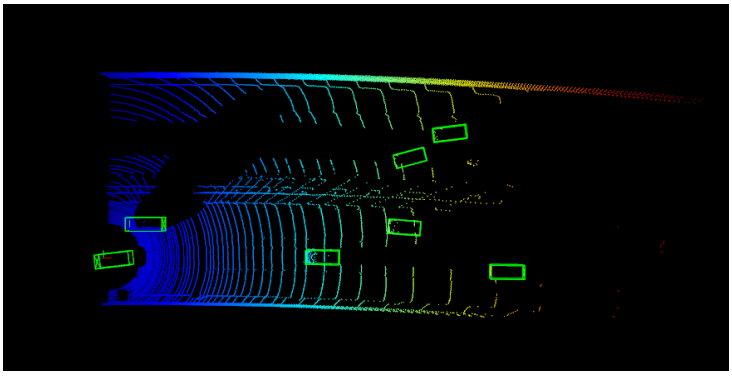


Figure 30: Prediction Bird Eye View LIDAR Test Dataset(000744)

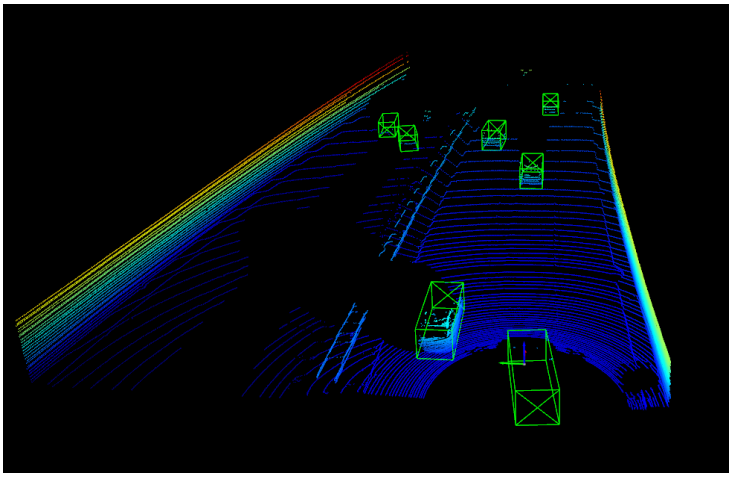


Figure 31: Prediction Bounding Box LIDAR Test Dataset(000744)

Figure 32: Sample Image Test Dataset(003698)

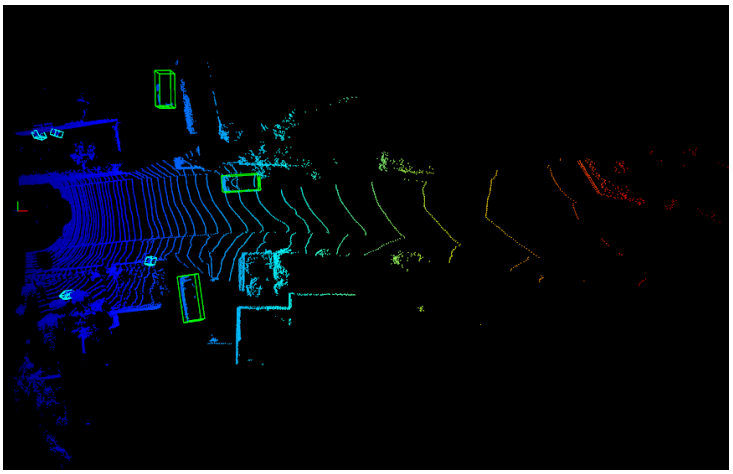


Figure 33: Prediction Bird Eye View LIDAR Test Dataset(003698)

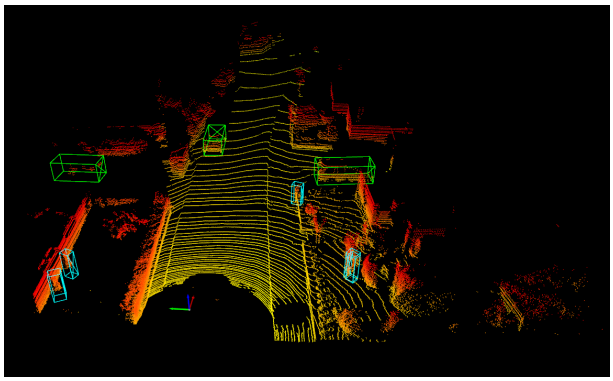


Figure 34: Prediction Bounding Box LIDAR Test Dataset(003698)

Figure 35: Sample Image Test Dataset(003640)

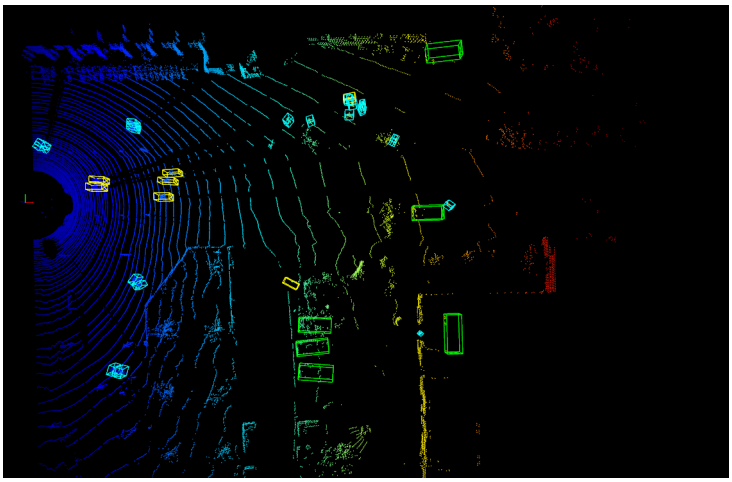


Figure 36: Prediction Bird Eye View LIDAR Test Dataset(003640)

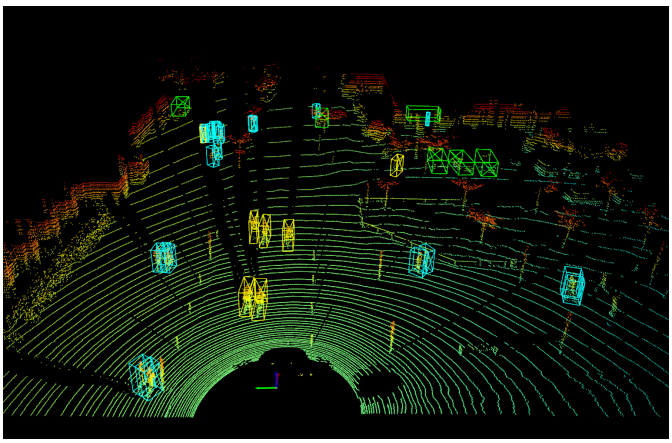


Figure 37: Prediction Bounding Box LIDAR Test Dataset(003640)

Figure 38: Sample Image Test Dataset(006258)

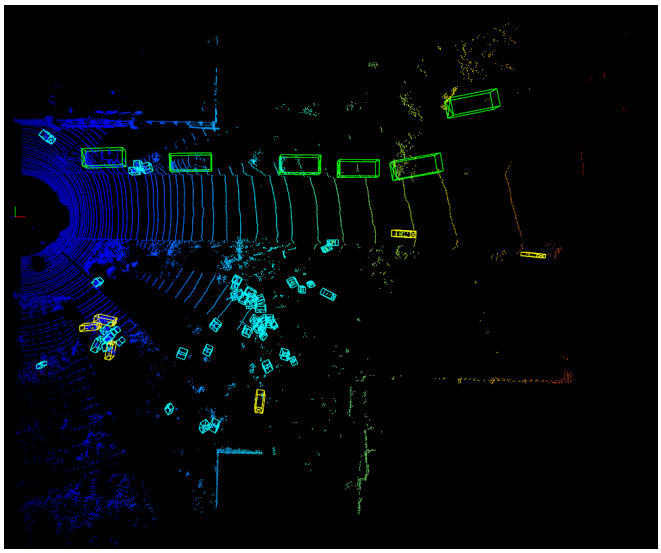


Figure 39: Prediction Bird Eye View LIDAR Test Dataset(006258)

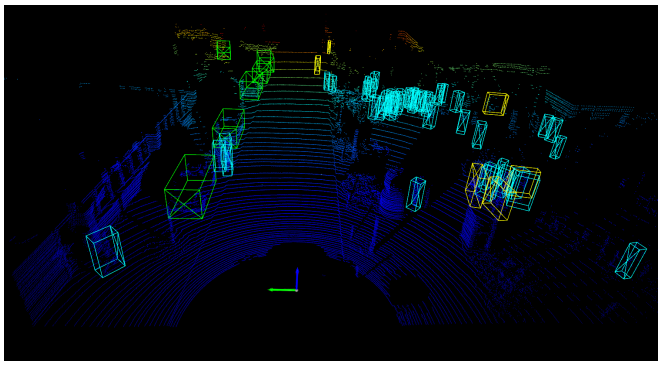


Figure 40: Prediction Bounding Box LIDAR Test Dataset(006258)

5.3 Challenges

The main challenge we faced was working with the full dataset of Nuscenes (v1.0 trainval). With an RTX3050 GPU equipped with 4GB of memory, we encountered limitations during training as our hardware proved insufficient for batch sizes of 4. Consequently, we reduced the batch size to 2 and trained only 1/10th of the full dataset for 5 epochs, which consumed approximately 8 hours. However, evaluating the results posed a challenge as the metadata information is provided for the complete dataset. Attempts to modify the metadata to suit our needs resulted in errors when using the nuScenes dev kit library for evaluation. Additionally, the entire LIDAR dataset requires approximately 280 GB of disk space, needing to do a re-partitioning of the PC's operating system. Despite these efforts, training remained unfeasible due to our limited single GPU memory.

Subsequently, we worked with the mini dataset and successfully trained it for 20 epochs, as described in the paper. While the resulting metrics for the three classes (Cars and Pedestrians) were satisfactory, the low performance of other classes led to an overall low mAP and NDS score.

6 Conclusion and Learning outcomes

In conclusion, our project focused on implementing the VoxelNext model using the NuScenes and KITTI datasets within the OpenPCdet open-source library for training and testing. While we encountered challenges with training the full version of the model, successful outcomes were achieved with the mini version. Working with the KITTI model, the model was successfully trained using the full dataset and the resulting metrics were found to be satisfactory compared with state of art models.

In summary, our journey through the project has been marked by overcoming challenges, particularly in handling sparse LiDAR data and ensuring accurate object detection. This experience provided us with valuable insights into the rapidly evolving field of 3D object detection, improving our technical skills and problem-solving abilities. We deepened our understanding of model architectures and their practical applications. Overall, this project was a rewarding learning experience that equipped us with valuable skills for future endeavors in computer vision and machine learning.

7 Future work

With a higher-end GPU memory capacity, it's possible to train the NuScenes full dataset and benchmark its results against other state-of-the-art models. Notably, the VoxelNext model [1], has demonstrated exceptional performance for object tracking. It presents an opportunity for further exploration and enhancement along with 3D box segment anything.

Regarding the KITTI dataset, there's potential to advance the current model's performance through the utilization of various data augmentation techniques during training. Furthermore, considering the success of the VoxelNext model, its integration for tracking purposes within the KITTI dataset requires investigation and implementation.

8 Github Repository Link

We provide the links for our code repository which contains training and evaluation pipelines, as well as our checkpoints for performing models that were mentioned in the Results section.

GitHub link: `https://github.com/Bibek199/VoxelNext_Projektarbeit`

We tried our best to explain the corresponding guidelines within the repository's README.md file. We share a step by step guide from setting up the OS environment to visualization of the results. We also share some insights on solving some problems we faced during the execution of the program.

The trained model checkpoints, log files and evaluation results are included inside the output folder of the repository.

Bibliography

[1] J. L. X. Z. X. Q. a. J. J. Yukang Chen, “VoxelNeXt: Fully Sparse VoxelNet for 3D Object Detection and Tracking,” 2023. [Online]. Available: https://arxiv.org/pdf/2303.11301.pdf.

[2] Y. W. a. J. Ye, “An Overview Of 3D Object Detection,” 2020. [Online]. Available: https://arxiv.org/pdf/2010.15614.pdf.

[3] Y. Z. a. O. Tuzel, “VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection,” 2017. [Online]. Available: https://arxiv.org/pdf/1711.06396.pdf.

[4] X. Z. a. P. K. Tianwei Yin, “Center-based 3D Object Detection and Tracking,” 2021. [Online]. Available: https://arxiv.org/pdf/2006.11275v2.pdf.

[5] “An Introduction to 3D Object Tracking (Advanced),” 2023. [Online]. Available: https://www.thinkautonomous.ai/blog/3d-object-tracking.

[6] Y. Massoud, “Sensor Fusion for 3D Object Detection for Autonomous Vehicles,” VIVA Research Lab, Otawwa, 2021.

[7] C. F. Z. C. F. Z. a. Y. X. Haozhe Qi, “P2B: Point-to-Box Network for 3D Object Tracking in Point Clouds,” 2020. [Online]. Available: https://arxiv.org/pdf/2005.13888.pdf.

[8] G. R. a. A. Patole, “End-to-End 3D Object Detection using LiDAR,” 2023. [Online]. Available: https://arxiv.org/pdf/2312.15377.pdf.

[9] R. T. O. S. H. P. N. D. F. H. M. K. a. D. H. (. C. Zijie J. Wang, “CNN EXPLAINER: Learning Convolutional Neural Networks with,” 2020. [Online]. Available: https://arxiv.org/pdf/2004.15004.pdf.

[10] G. Z. J. X. L. L. C. J. S. X. Z. W. Ziying Song, “VoxelNextFusion: A Simple, Unified and Effective Voxel Fusion Framework for Multi-Modal 3D Object Detection,” 2024. [Online]. Available: https://arxiv.org/pdf/2401.02702.pdf.

[11] K. O. a. R. Nash, “An Introduction to Convolutional Neural Networks,” 2015. [Online]. Available: https://arxiv.org/pdf/1511.08458.pdf.

[12] “Kitti dataset,” [Online]. Available: https://www.cvlibs.net/datasets/kitti-360/.

[13] P. L. C. S. a. R. U. A Geiger, “Vision meets robotics: The KITTI dataset,” 2013. [Online]. Available: https://doi.org/10.1177/0278364913491297.

[14] D. C. B. a. M. Y. Hiep Anh Hoang, “TSSTDet: Transformation-based 3-D Object,” IEEE SENSORS JOURNAL, 2023.

[15] S. V. H. C. L. Z. Y. a. O. B. Alex H. Lang, “Pointpillars: Fast encoders for object detection from point clouds,” 2019. [Online]. Available: https://arxiv.org/pdf/1812.05784.pdf.

[16] C. G. L. J. W. S. X. W. H. L. Shaoshuai Shi, “PV-RCNN: Point-Voxel Feature Set Abstraction for 3D Object Detection,” 2021. [Online]. Available: https://arxiv.org/pdf/1912.13192.pdf.

[17] W. T. L. J. a. C.-W. F. Wu Zheng, “SE-SSD: Self-Ensembling Single-Stage Object Detector From Point Cloud,” 2021. [Online]. Available: https://arxiv.org/pdf/2104.09804.pdf.

[18] Q. Z. Z. Z. J. H. a. Y. Y. Yifan Zhang, “GLENet: Boosting 3D Object Detectors with Generative Label Uncertainty,” 2023. [Online]. Available: https://arxiv.org/pdf/2207.02466.pdf.

[19] Y. Z. a. U. N. Qiangeng Xu, “Behind the Curtain: Learning Occluded Shapes for 3D Object Detection,” 2021. [Online]. Available: https://arxiv.org/pdf/2112.02205.pdf.

List of Figures

List of Tables