The Development of Grid-based 3D Object Detectors: A Comprehensive Survey

Haoquan Zhang
School of Future Technology
South China University of Technology

Wei Wu
School of Future Technology
South China University of Technology

Zhizhan Zhang
School of Future Technology
South China University of Technology

Shuhong Liu
School of Future Technology
South China University of Technology

Abstract

Machine vision is an essential component of intelligent agents' perception of the outside world. Compared to 2D visual perception, 3D vision can more directly obtain the size and position of external objects. However, the development of 3D vision is not as extensive as that of 2D vision, which has been developed for many years. It can be said that 3D vision is one of the most important problems in computer vision today. If we can transfer the widely used methods of 2D vision to the processing of 3D vision, then its development will inevitably be more rapidand efficient. We found that the grid-based approach makes the transfer of 2D methods possible when we extensively read papers. This paper summarizes the development process and differences of some grid-based 3D object detectors, and analyzes them around the two core methods of visual processing: CNN and transformer[43].

1 Introdunction

3D vision is the ability to perceive and understand the three-dimensional structure of the world from images or videos. It is a challenging and important problem in computer vision, as it enables many applications that can benefit various sectors of society. For example, 3D vision can help autonomous vehicles navigate safely and efficiently on complex roads, medical diagnosis become more accurate and efficient with interactive 3D visualization of X-rays, MRI, and CT scans, crop monitoring improve agricultural productivity and sustainability, and defect detection enhance quality control in manufacturing. These are just some of the examples of how 3D vision can have a positive impact on real-world problems and improve human lives.

3D vision and 2D vision are two ways of perceiving and understanding the three-dimensional structure of the world from images or videos. 3D vision can capture or estimate the depth information of the scene, which is the distance of each point from the camera. 2D vision can only obtain the intensity or color information of each pixel, without knowing how far away they are. Therefore, 3D vision can provide a more complete and accurate representation of the scene than 2D vision, which can only provide a flat projection. 3D vision has many advantages over 2D vision in various applications that require depth information, such as object recognition, pose estimation, scene reconstruction, autonomous navigation, and so on. 3D vision can also overcome some of the limitations and challenges of 2D vision, such as occlusions, illumination changes, lack of contrast, and perspective distortions. These factors can affect the quality and reliability of 2D images, but they have little impact on 3D depth data. Therefore, 3D vision is a more robust and versatile way of seeing and interpreting the world than 2D vision.

Preprint. Under review.

Since 2016, deep learning has become more mature in 3D vision applications. In order to directly transfer the 2D object detection algorithms based on CNN, researchers came up with two forms of front view (FV) and bird-eye view (BEV) to convert the three-dimensional discrete point cloud data into 2D image data, and some original LiDAR data information was encoded as features corresponding to pixels. This operation can well transfer the object detection work on 2D. Some researchers also converted the image into a three-dimensional voxel grid and used 3D CNN to extract features from it. In 2017, the transformer model was released. Due to its excellent global analysis ability, the transformer achieved good results on large datasets, and the computer vision academic community began to develop vision models based on the transformer model. However, due to its expensive training cost, many researchers also combined CNN and transformer models, and did self-attention under the multi-dimensional feature map of CNN, combining the advantages of CNN being easy to train and transformer being able to dynamically process data. They achieved good results on some smaller datasets.

The structure of our article will explain the development of 3D vision detectors in recent years, and will be divided into the following parts:

1. Some background of 3D vision, including point cloud data gridization preprocessing methods and commonly used datasets.
2. Introduction of CNN model principle and detector based on CNN.
3. Introduction of transformer model and detector based on transformer model.
4. Comparison and analysis of CNN and transformer models from the perspective of inductive bias

2 Background

2.1 Gridization Processing

Gridization is a common data preprocessing method for 3D vision tasks, because grids have better applications than raw point cloud data. Voxelization is a popular processing method, in which point cloud data is discretized into square grids of a certain resolution in the Vote3D[48] method, and detectors such as Vote3Deep[10] and 3DFCN[27] also use this discretized representation. However, not all gridization processes have equal length, width and height. The SECOND[44] method uses a voxel size of vD = 0.4 × vH = 0.2 × vW = 0.2 m. In addition, methods such as PointPillars[26] and BirdNet[3] discretize the x and y dimensions into squares, but do not discretize the z axis. PointPillars groups point cloud data according to Pillars, and then processes the point cloud data on the grouped scale; BirdNet, as a representative of the bird's eye view (BEV) processing method, encodes the height information of the point cloud data into the flattened pixels according to a certain method, and processes it as a channel. The superiority of BEV is that its data format is the same as that of traditional images, so many existing 2D detection models can be easily transferred into 3D tasks.

2.2 3D Bounding Box and Orientation Encoding

2.2.1 8-Corner Encoding

VeloFCN[28] adopts a natural way to represent the bounding box. It proposes to use the coordinates of the eight corners of the box to represent the bounding box. If we use c_i to denote a corner, then the coordinates of c_i are (x_i, y_i, z_i), and the total coordinate representation is $(c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8)$. Therefore, this version of encoding requires 3 × 8, that is, a total of 24 coordinate information to represent the bounding box. Considering the shapes of vehicles, bicycles and pedestrians in reality, the direction of the bounding box is directly set to the direction corresponding to the longest edge. But this scheme also has some irrationality. Bicycles and vehicles seem to fit this longest edge setting better, but pedestrians do not conform to this assumption.

2.2.2 4-Corner 2-Height Encoding

AVOD[25] improves on VeloFCN's method and combines the actual road scene to design the bounding box encoding as $(c_1, c_2, c_3, c_4, h_1, h_2)$, which reduces the encoding length from 24 to 14. Among them, c_i represents corner points, h_1 represents the distance from the bounding box to the ground,

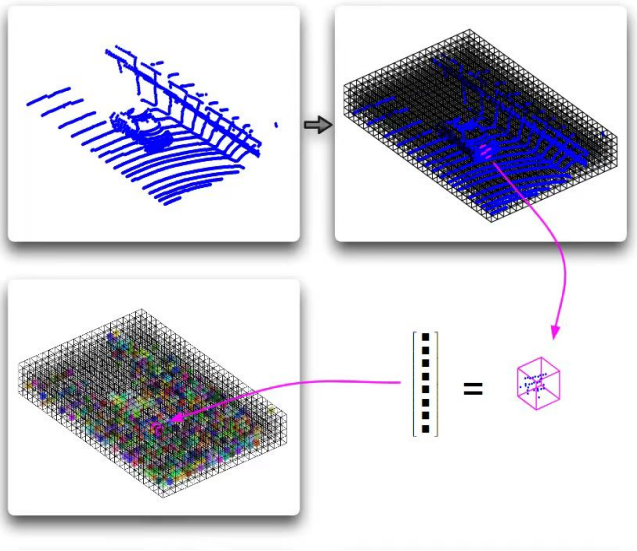

Figure 1: Illustration of gridization processing and feature encoding in Vote3D Detector

and h_2 represents the height of the bounding box. This design is consistent with reality. After all, vehicles, bicycles and pedestrians are basically equal in width and length up and down, so VeloFCN's 8-corner point encoding method can naturally be reduced to 4-corner points. And AVOD also solves the problem of direction orientation to a certain extent. For example, suppose we have an object moving in the BEV space at an angle $\theta = \pi/2$. We can use $(x_\theta, y_\theta) = (\cos(\frac{\pi}{2}), \sin(\frac{\pi}{2})) = (0, 1)$to represent this direction vector. This vector points to the positive direction of the y-axis in the BEV space, which makes it easy to understand the direction of motion of this object. If the angle θ of this object exceeds the range of [$-\pi$, π], we can still use this method to represent its direction because this method implicitly handles the angle wrapping problem.

2.2.3 Center-Size and Axis Aligned Encoding

Although AVOD's encoding is already very concise, in the paper [40], the author proposes a method different from 8-Corner Encoding and 4-Corner 2-Height Encoding. The author uses the center of the bottom surface and the length, width and height of the bounding box to encode information, that is, (t, d_x, d_y, d_z), where t is the center coordinate of the ground, and d_x, d_y, d_z are the distances of the object in the x, y, and z three directions respectively. This encoding length is only 6, and it also conforms to the actual situation. But such a bounding box is along the preset direction axis, and it can hardly represent the real edge and direction of the vehicle.

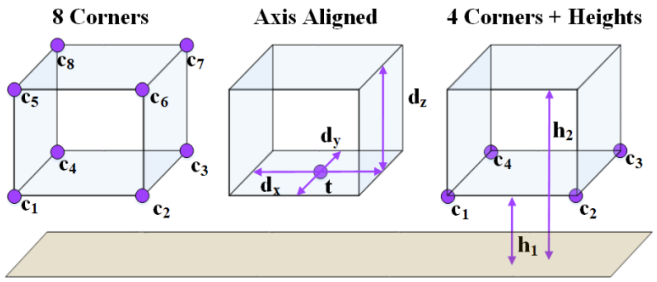

Figure 2: A visual comparison between the 8 corner box encoding proposed in [28], the axis aligned box encoding proposed in [40], and 4 corner encoding in [25].

2.3 Datasets

2.3.1 PASCAL VOC

PASCAL VOC itself is a computer vision challenge, and the PASCAL VOC dataset provided here is a pioneer for object detection or segmentation types, and it is still used by a large number of papers for training and detection. The two most commonly used data sets are PASCAL VOC2007[11] and PASCAL VOC2012[12], but these two data sets are incompatible. Among them, VOC2007 contains about 10,000 images, and the proportion of training sets and test sets is about 50%. VOC2012 contains all the data from 2008 to 2011, with about 12,000 images and about 30,000 detection objects. The PASCAL VOC competition is no longer running, but researchers can still use these datasets and upload predictions for evaluation.

2.3.2 Waymo

While reading the papers, we found that many papers use Waymo[41] data sets for training and testing their 3D algorithms besides using KITTY data sets, so here is a brief introduction to Waymo data sets. The Waymo dataset, released by Google in 2020, is the largest public self-driving data set to date. Waymo's data set, collected datas by using five Lidar sensors and five high-resolution pinhole cameras. It contains more than 3,000 driving clips, each of which is about 20 seconds of continuous driving footage, covering cities such as San Francisco, Phoenix and Mountain View. So different information from different regions will be well represented in Waymo's data set. The Waymo dataset provides 3D ground boundary truth frames, 2D tightly fitted boundary frames of camera images, and very large lidar box and camera case annotations for researchers and practitioners to study. The Waymo dataset is also much more plentiful than the KITTY dataset, containing different information for different weather conditions, different time periods, and so on. As a result, Waymo's data set has been used extensively to evaluate and compare 3D algorithms since its release.

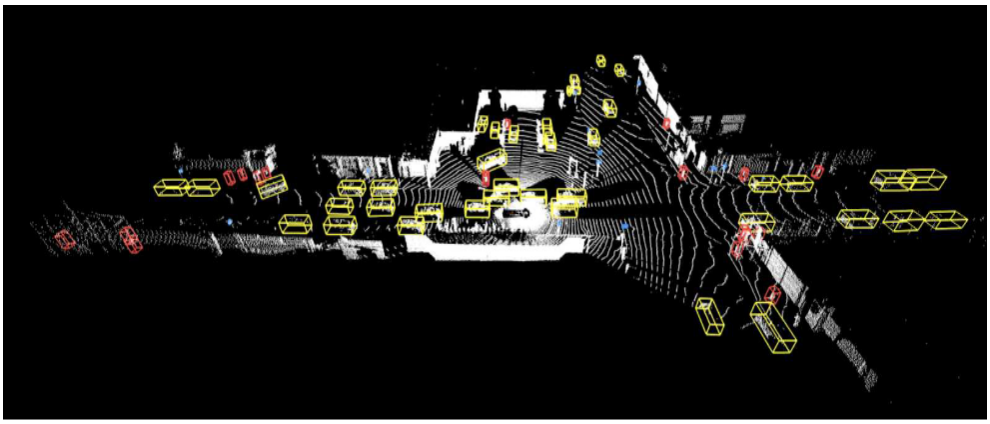

Figure 3: A LiDAR label example of Waymo dataset.

2.3.3 KITTI

The KITTI[13] dataset is a collection of real-world computer vision benchmarks for autonomous driving. It includes tasks such as stereo, optical flow, visual odometry, 3D object detection and 3D tracking. The dataset was captured by a station wagon equipped with two color and grayscale cameras, a 3D laser scanner and a GPS system. The dataset covers various traffic scenarios in the city of Karlsruhe, Germany, as well as rural areas and highways. The dataset provides accurate ground truth annotations for 3D bounding boxes, as well as calibration parameters and camera poses. The KITTI dataset is widely used by researchers and practitioners to evaluate and compare their methods for 3D vision.

Many 3D detectors use the KITTI dataset to train their models because it is a well-established benchmark that allows them to compare their performance with other methods. The KITTI dataset also covers a wide range of scenarios and object classes, such as cars, pedestrians, cyclists, etc., which makes it suitable for generalizing to real-world applications. Additionally, the KITTI dataset

provides both LiDAR and stereo data, which can be fused to improve the accuracy and robustness of 3D detection.

2.3.4 ApolloScape

The ApolloScape[22] Dataset for Autonomous Driving is a large-scale and comprehensive dataset that aims to facilitate research and innovation in various aspects of autonomous driving, such as perception, navigation, control, and simulation. It is part of the Apollo project, an open platform for self-driving cars developed by Baidu. The dataset contains rich and diverse information that is essential for understanding complex urban traffic scenarios, such as:

1. Dense semantic 3D point cloud for the environment (20+ driving sites)
2. Stereo driving videos (100+ hours)
3. High accurate 6DoF camera pose (translation $\leq$ 50mm, rotation $\leq 0.015\ ^\circ$)
4. Videos at same site under different day times (morning, noon, night)
5. Dense per-pixel per-frame semantic labelling (35 classes, 144K+ images)
6. Per-pixel lanemark labelling (35 classes, 160K+ images)
7. Semantic 2D instances segmentation (8 classes, 90K+ images)
8. 2D car keypoints and 3D car instance labelling (70K cars)

The dataset also provides several standard benchmarks for scene parsing , instance segmentation , lanemark parsing , self-localization , and other tasks. The dataset is collected using a high-end acquisition system that consists of two laser scanners, two front cameras, and a measuring head with IMU/GNSS. The dataset is still growing and evolving, and will soon include new tasks such as 3D car instance shape and pose, 3D car tracking, etc. The dataset covers multiple cities and sites in China under various driving conditions. The dataset and its toolkit are publicly available at `https://apolloscape.auto/`

2.3.5 nuScenes

NuScenes[5] is a public large-scale dataset for autonomous driving that enables researchers to study challenging urban driving situations using the full sensor suite of a real self-driving car. It consists of 1000 scenes, each 20 seconds long and fully annotated with 3D bounding boxes for 23 classes and 8 attributes. It covers two diverse cities: Boston and Singapore, with different weather and light conditions, traffic rules, and object locations. It is the first dataset to provide 360-degree sensor coverage from the entire sensor suite, including 6 cameras, 5 radars and 1 lidar . It is also the first autonomous driving dataset to include radar data and data from nighttime and rainy conditions. nuScenes is a holistic scene understanding benchmark for autonomous driving that supports multiple tasks such as object detection, tracking, behavior modeling, and map learning .

The nuScenes dataset is inspired by the pioneering KITTI dataset, but it has several advantages over it. Compared to KITTI, nuScenes includes 7x more object annotations and 100x more images. It also has a more diverse and realistic set of scenarios, such as left versus right hand traffic, construction vehicles, traffic cones and barriers . Moreover, nuScenes provides low-level data from the vehicle's CAN bus, such as steering angle, brake pressure and vehicle speed. Additionally, nuScenes shares HD internal semantic maps with 11 layers, such as drivable area, walkway and crosswalk.

3 Detectors Using CNN(Convolutional Neural Network)

3.1 Convolutional Neural Network Theory

3.1.1 What Is CNN

Convolutional neural network (CNN) is a deep learning model that can handle images and any data that can be transformed into a similar image structure. The characteristic of convolutional neural network is to use convolution calculation to extract local features of input data, thus achieving efficient representation and classification of data. Convolutional neural network consists of multiple layers,

including input layer, convolution layer, activation layer, pooling layer, fully connected layer, etc. Its multi-layer structure introduces a large number of learnable parameters, which makes CNN model have extremely strong learning ability, and its parameter sharing property of convolution kernel makes CNN have good generalization ability and robustness. The core of CNN lies in its convolution layer. The mainstream 2d-CNNs use 3×3 convolution kernels, and the feature maps of different scales obtained after multiple convolutions also contain semantic information and texture features of different dimensions. Researchers will use different levels of feature maps and different levels of feature fusion to deal with different tasks.

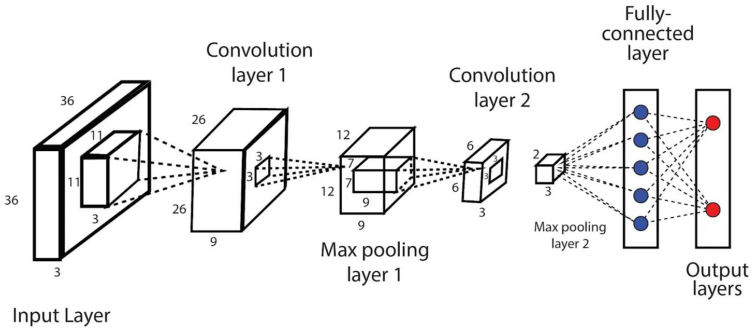

Figure 4: An example of the architecture of CNN.

3.1.2 Why We Use CNN

Thanks to the proposal of resnet[21], the depth of CNN has been improved, which makes the extracted features more accurate and rich in semantic information. And the emergence of FPN[29] method can also solve the problem of small object feature disappearance in CNN high-dimensional feature map to a certain extent. CNN combines various methods, as well as its own powerful feature extraction ability, before the transformer is proposed, it is the basis of sota model on most image problem benchmarks, even after the transformer is proposed, CNN can still have an advantage in industrial applications with its own low overhead. Today, when 2d image processing is gradually improving, 3d problems are also very natural and reasonable choices for the dimensional application of 2d models.

3.2 Detectors Using CNN

3.2.1 Detectors of directly using 3D CNN after voxelization

Vote3D[48] is an early grid-based detector. The insight of this paper is that: compared to the 2D information which is very dense, the 3D information is sparser, so the authors only perform computations on the voxels that have content, greatly reducing the computational cost. The first step is the encoding of the three-dimensional image. Given a grid size, the data related to the point cloud in the cell is quantized into a vector representing the voxel. If a grid has no points, it is set to zero. The encoding vector has 6 parameters, which are binary occupancy value, a mean of reflectance, a variance of reflectance and three shape factors related to the scattering of points (polarization factor, orientation factor and phase function). Vote3D uses a sliding window to find objects. The sliding window starts from a corner, and all the encoded feature vectors in the window are stacked up into a long vector after voting calculation. Then they are sent to a support vector machine (SVM) classifier for clustering, and then regressed to bounding boxes by separately trained networks. However, non-maximum suppression (NMS)[33] is needed to remove redundant candidate boxes. Vote3Deep[10] builds on Vote3D by using the equivalence of convolution and voting in sparse point clouds to extract features from three-dimensional grid data. Vote3Deep also applies L1 regularization to the intermediate layer and uses relu as the activation function. These operations enable the convolutional layer to maintain the sparsity of the image to some extent.

3DFCN[27] transfers the fully convolutional network (FCN) to 3D, detecting and localizing objects in point clouds with 3D boxes. FCN is a popular end-to-end object detection framework that has the best performance on tasks including ImageNet, Kit, ICDAR and others. The authors naturally

extend the mechanism of 2D CNN to 3D on cubic grids. The network structure of 3DFCN has three convolutional layers conv1, conv2 and conv3 that successively downsample the input image by $1/2^3$. The layers deconv4a and deconv4b respectively upsample the incoming image by 2^3. The deconv4a and deconv4b layers are used to output the object map and the bounding box map respectively.

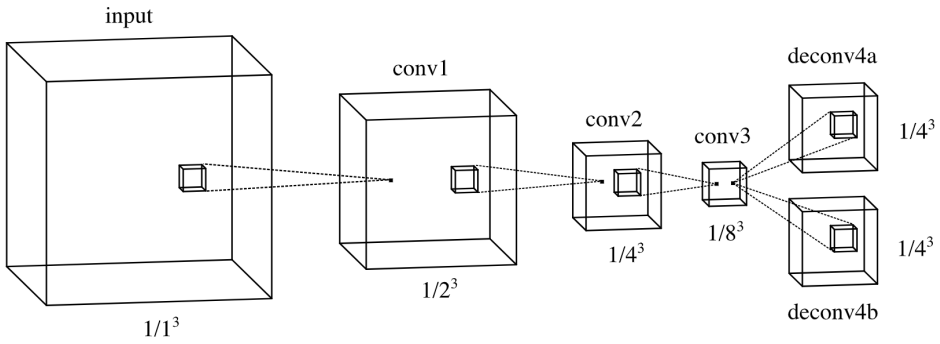

Figure 5: A sample illustration of the 3D FCN structure.

3.2.2 Detectors of grouping and processing point cloud data after gridization

PointPillars[26] proposed a novel point cloud encoding method, which was implemented by Pillar Feature Net. After the original point cloud was discretized on the X-Y plane, many pillars were generated, which were called pillars. The points in the pillars were all augmented with data. The augmentation method was to add xc, yc, zc, xp and yp to the points, where the subscript c represented the arithmetic mean of all the points in the pillars, and p represented the offset of each point relative to the arithmetic mean. In this way, the augmented points had a 9-dimensional representation. Through this representation, the point cloud could be expressed as a tensor of size (D, P, N), where D was the dimension of the point cloud, P was the pillar index, and N was the number of points in the pillar. Due to the limitation of N, when there were too many points in a pillar, the points were randomly sampled; when there were too few points in a pillar, the empty parts were filled with 0. Then the authors used a simplified PointNet[35] to generate a vector of size (C, P, N), which had only one BatchNorm[23] layer, one ReLU[32] layer and one linear layer. Then a max operation on the feature channels was performed to obtain an image of size (C, P). In fact, an index corresponded to a coordinate on a two-dimensional plane, so the final feature image was scattered back to its original position, and finally a pseudo-image of size (C, H, W) was obtained. This was part of Pillar Feature Net. The backbone after this was a 2D CNN, and finally a Detection Head using the Single Shot Detector (SSD)[30] was used to perform 3D object detection.

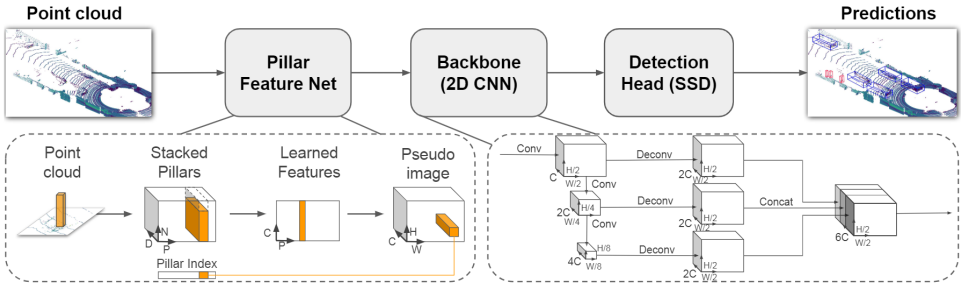

Figure 6: The main components of the network are a Pillar Feature Network, Backbone, and SSD Detection Head[26].

VoxelNet[51] proposed an End-to-End Learning for Point Cloud Based 3D Object Detection method. VoxelNet divided the point cloud into equidistant 3D voxels and transformed a group of points within each voxel into a unified feature representation through a newly introduced voxel feature encoding (VFE) layer. In this way, the point cloud was encoded into a descriptive volumetric representation,

which was then connected to RPN [37] to generate detections. The feature learning network took the raw point cloud as input, partitioned the space into voxels, and converted the points within each voxel into vector representations that represent shape information through VFE. The voxel feature encoding layer first received point-wise input, obtained point-wise features after a fully connected layer, and then obtained locally aggregated features through element-wise maxpooling. Finally, locally aggregated features were concatenated with point-wise features to obtain point-wise concatenated features. In fact, the feature learning layer used multiple VFE layers to learn the features of the points in the voxel, and finally obtained voxel-wise features through a fully connected layer and an element-wise maxpool layer. Through the above operations, the space was represented as a sparse 4D tensor. Subsequently, the convolutional intermediate layer processed the 4D tensor to aggregate spatial context. Finally, RPN generated 3D detections. Experiments on the KITTI car detection benchmark showed that VoxelNet significantly outperformed state-of-the-art lidar-based 3D detection methods.

One point to note is that the voxelization method of VoxelNet imposes a limit on the number of points in each voxel, which means that when the original data has more points than the limit, the points are randomly sampled to the limit number. This is also called static voxelization. MVF[50] improves on the static voxelization method of VoxelNet by using a dynamic voxelization approach, which can preserve almost all the point information and voxel information, rather than potentially losing key information due to random sampling like VoxelNet.

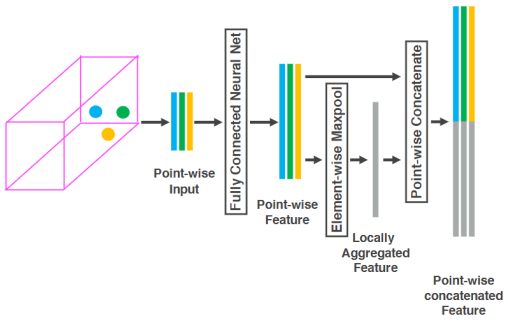

Figure 7: A sample illustration of Voxel feature encoding layer.

Coincidentally, the structure of the SECOND[44] method is highly similar to that of VoxelNet. The SECOND detector consists of three parts: (1) voxel feature extractor; (2) sparse convolutional intermediate layer; (3) RPN. First, in the data preprocessing stage, for all tasks, the authors used vD = 0.4 × vH = 0.2 × vW = 0.2 m as the voxel size. The maximum number of points in each empty voxel used for car detection was set to T = 35, which was chosen based on the distribution of the number of points per voxel in the ki dataset; the corresponding maximum value for pedestrian and cyclist detection was set to T = 45, because pedestrians and cyclists are relatively small, so voxel feature extraction requires more points. The benefit of setting the maximum point density is that there is a problem of uneven sampling rate in the three-dimensional point cloud data. Due to the limitations of the radar equipment, the sampling point density in the distance is much smaller than that in the near distance. When setting the maximum point density, the density difference can be eliminated to some extent, making the model more generalizable. The Voxelwise Feature Extractor used by the SECOND method used the VFE structure proposed in VoxelNet, and the features of the voxels were extracted after several VFE layers and an FCN layer. In the sparse convolutional layer, as an alternative to regular sparse convolution, submanifold convolution[16] restricts output positions to be active only when and only when the corresponding input positions are active. This avoids generating too many active positions, which may lead to a slowdown in subsequent convolutional layers due to a large number of active points. Finally, RPN obtained detection results.

In 2020, HVNet[47] proposed AVFE and AVFEO layers based on VFE. In the first part of the HVNet architecture, hybrid voxel feature extractor (HVFE), the authors used a multi-scale discretization encoding scheme to enable the extractor to obtain Multi-scale Pseudo-Image Feature. In this step, the point cloud is first discretized into different scales, denoted as s_t, and the points are assigned an index, denoted as $\mathbf{c}^{s_t}$. According to PointNet[35], each point with an index is encoded with a feature denoted as $\mathbf{F}^{s_t}$. Then, an attention mechanism is used to obtain $\mathbf{G}^{s_t}$ with the same dimension as

$\mathbf{F}^{s_t}$. After passing through the AVFE layer, the final point-wise feature is obtained by point-wise concatenation. This point-wise feature can be converted into Scale s_r Pseudo-Image Feature by the AVFEO layer at a specific scale (denoted as s_r). The subsequent architecture uses a 2d FPN structure to obtain a feature map that combines low-dimensional and high-dimensional features, which is then used for detection. Compared with VFE in VoxelNet[51], AVFE and AVFEO layers proposed by HVNet[47] based on VFE can better aggregate point cloud features at different scales, and can flexibly output Pseudo-Image Feature at any desired scale.

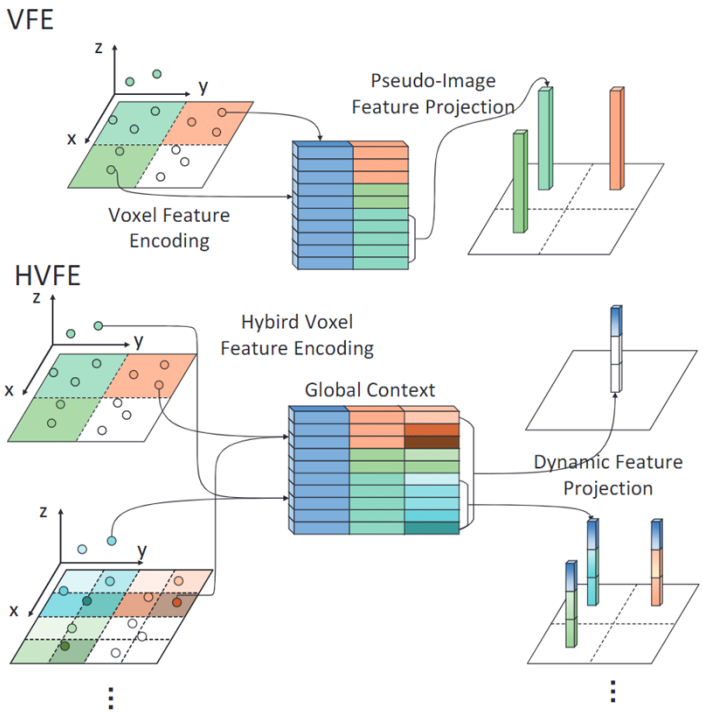

Figure 8: The voxel feature extraction strategy of VFE methods and HVNet. Each point feature in VFE methods only contains one specified scale knowledge. As to HVNet, point features under hybrid scales are aggregated into a global context, then projected to features in dynamic target scales[47].

3.2.3 Detectors using BEV preprocessing and 2D CNN

BEV is a way of projecting point cloud data onto the XY plane, transforming the three-dimensional point cloud data into two-dimensional image data. After filtering, cropping, downsampling and other operations on the raw lidar data, the point cloud is discretized. The point cloud data is divided into several voxels on the XY plane, and each voxel's point cloud data is represented by a feature vector. The feature vector has different encoding methods in different detectors. Lidar data contains the position information and reflectance data of the points, and many attributes can be derived from this for data augmentation, such as the average position information (or centroid) of the points in the voxel, the distance between the points and the lidar acquisition device, and the offset between the centroid and the voxel center. The feature encoding or data embedding method of the bev method can be a subset of these features. In fact, BEV images are two-dimensional, which means that their height information is not directly expressed in the tensor, but is encoded into the original features through some mapping. But one of the prior knowledge, or bias induction of road images is that roads generally do not have too much height difference, so the loss of height information is acceptable, and it does not have much impact on object detection and bounding box regression.Compared with the mapping method of FV image, the mapping method of BEV does not produce occlusion of objects, which is helpful for regressing bounding box, and the mapping form of BEV is helpful to obtain dense image, rather than sparse image like 3D voxel image.

This paper proposes a BirdNet[3]-based BEV image encoding method that uses height, intensity and density as features to represent lidar data. Height represents the highest point in each cell (no higher than 3m); intensity represents the average value of intensity of reflectance of all points; density represents the degree of density of points, calculated as the ratio of the number of points actually existing in a unit to the maximum possible number of points. To eliminate the differences between different lidar devices, this paper standardizes the lidar data into individual entities with a size of δ × δ and a height of H_{top}. And these entities have the three parameters mentioned in the previous paragraph. Because the lidar rays rotate to scan objects, each square is cut by the circle of the lidar scan from a bird's eye view. There are three situations:

1. Every corner of the square is inside the circle
2. Every corner of the square is outside the circle
3. The square and the circle are cut, and intersect at two points

According to this formula, the intersection point P can be calculated. Since V has two candidate points, there will actually be two possible P points.

$$\begin{cases} P_x^2 + P_y^2 = d^2 \\ P_y - C_y = \frac{V_y - C_y}{V_x - C_x}\left(P_x - C_x\right), V \in \{B, D\} \end{cases}$$

Using the arctan formula, the angles of the two cutting points can be calculated. According to the angular resolution $\delta\theta$ and the number of layers Np of the lidar, the maximum number of points in a cell can be calculated. Thus, the maximum possible number of points in a cell $M_{i,j_{\max}}$ can be obtained, and then the density can be calculated.

$$M_{p_{i,j}} = \begin{cases} \left\lceil \frac{|\theta_n - \theta_0|}{\Delta\theta} \right\rceil & \text{if plane intersects cell,} \\ 0 & \text{otherwise.} \end{cases}$$

$$M_{i,j_{\max}} = \sum_{p=0}^{N_p} M_{p_{i,j}}$$

In the object detection module, the authors mention that although Faster RCNN[37] is a structure designed for RGB channel 2D images, it can actually be used for any 2D structure, such as BEV images, which is actually a general structure. The paper uses VGG-16[39] as a feature extractor. But the authors found that after processing by the final convolutional layer, the image was 16 times smaller than the original image, which was not suitable for finding entities that were only a few pixels in size in BEV images, such as cyclists and pedestrians. Then they removed the fourth max-poling layer to reduce the degree of downsampling. They also used ROIAlign (ROI[19], Region of Interest) method to do feature pooling. The paper uses KITTI dataset, according to the annotation of the dataset, the authors generated data rotated by 90 degrees, 180 degrees, 270 degrees, so that the model's ability is not limited to the camera's field of view. They also flipped data, which can also enhance model performance.

Following BirdNet, PIXOR[46] and HDNET[45] detectors both adopted the BEV mapping scheme. PIXOR method did not use the common representation of occupancy, intensity (reflectance), density, and height feature[7]. Instead, for simplicity, it only used occupancy and intensity as the features. The authors computed the occupancy feature at a grid resolution of dL × dW × dH and added two extra occupancy feature layers to indicate out-of-range points. The intensity feature map was obtained at a grid resolution of dL × dW × H. Therefore, the final feature layer had H/dH + 3 channels. HDNET followed the encoding scheme of PIXOR, but differed from PIXOR in that it obtained a road mask that represented the road features from a high-resolution 2D image and concatenated it with the original features as the network input. In fact, the road mask helped to cluster objects by type.

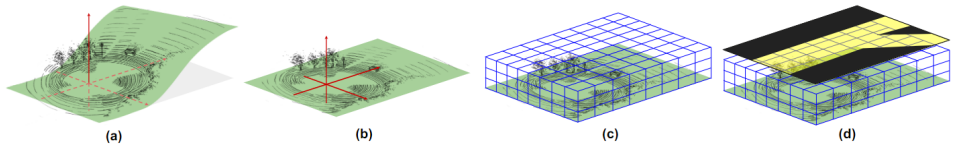

Figure 9: BEV LiDAR representation that exploits geometric and semantic HD map information.[45]

3.3 R-CNN Series Detectors

3.3.1 R-CNN Detector

R-CNN[15] detector is a detection algorithm that was published in 2014, which utilizes CNN convolutional neural networks to extend the excellent performance of CNN for image classification to the field of object detection. At that time, CNN was more used for image classification rather than object detection, and in order to better locate the target, this algorithm only used CNN algorithm to extract regional features, and used support vector machine for classification operation separately. As shown in Figure 10, using the existing selective search algorithm to extract about 2000 regions, then scaling each region to a uniform size through warp, and putting it into CNN convolutional neural network for feature extraction, after extracting about 4k-dimensional features, it is put into support vector machine for binary classification to determine the specific category that the feature belongs to. After extracting the required feature information through CNN convolutional neural network, it is necessary to use support vector machine[9][4][42] (SVM) to predict each feature category. After completing the classification prediction for each pre-selected region, all target boxes can be obtained. After SVM classification, we need to use non-maximum suppression algorithm to eliminate redundant boxes in the target boxes. Finally, in order to improve the accuracy of the target box, bounding-box regression was trained separately to improve accuracy.

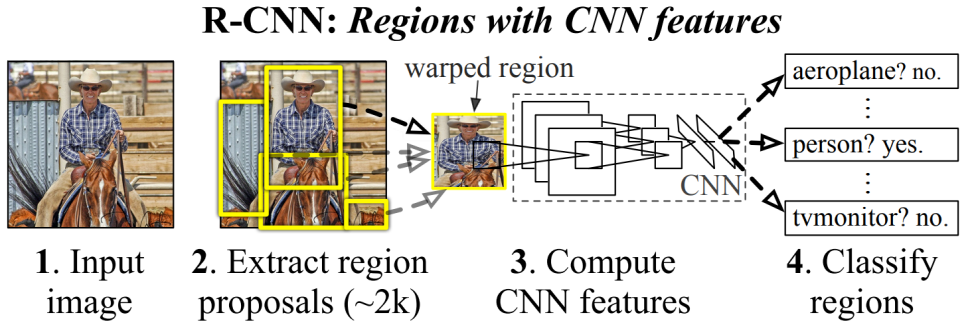

Figure 10: Object detection system overview of R-CNN[15]

R-CNN has a low feature dimension of only 4k when extracting features, which is significantly lower than other object detection methods. Moreover, it improves the efficiency of classification computation by merging each class and calculating the product of matrices. However, R-CNN requires many steps, including pre-selecting regions, training CNN convolutional neural network, training support vector machine SVM and training regression algorithm, which makes the training time very long and also occupies unnecessary disk space to cache the obtained features. In addition, the algorithm needs to perform a convolution for each pre-classified region, which involves a lot of unnecessary computation. Furthermore, even for testing, it needs to perform region pre-selection, CNN convolution and SVM classification operations for each test data, so the testing time is also very long. Therefore, even though R-CNN was more efficient than traditional object detection methods at that time, this algorithm still has a lot of room for improvement.

3.3.2 Fast R-CNN Detector

As introduced in section 3.3.1 regarding R-CNN, there is still room for improvement in the efficiency of R-CNN. The Fast R-CNN[14] algorithm, published in 2015, is an improvement based on the original R-CNN algorithm. Compared to R-CNN, Fast R-CNN simplifies the multi-stage training process into a single-stage process and updates all convolutional layers during training. Additionally, since Fast R-CNN uses a single-stage training process, it does not need to cache extracted features like R-CNN, greatly reducing disk usage. Given a complete image, the Fast R-CNN structure involves placing the entire image into a deep CNN convolutional neural network to obtain the features of the entire image. Then, based on RoI projection, candidate region features are obtained. The RoI pooling layer is used to convert non-fixed candidate region feature sizes into uniform sizes for output. The softmax function is used to predict feature categories, outputting k+1 categories (feature categories + background categories), while bounding-box regression is used to predict the coordinates of the box.

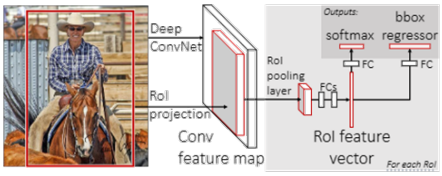

Figure 11: Structure of Fast R-CNN

The RoI pooling layer used in Fast R-CNN is similar to the SPP[20] spatial pyramid pooling layer used in the SPPnet network, which is also based on R-CNN. Both can convert the fixed input size of R-CNN into any size input. It limits the output size and dynamically changes the pool box size and stride based on the output size. Therefore, following the idea of SPPnet spatial pyramid pooling, the entire image can be placed into a CNN neural network without the need for warp scaling to the same size.

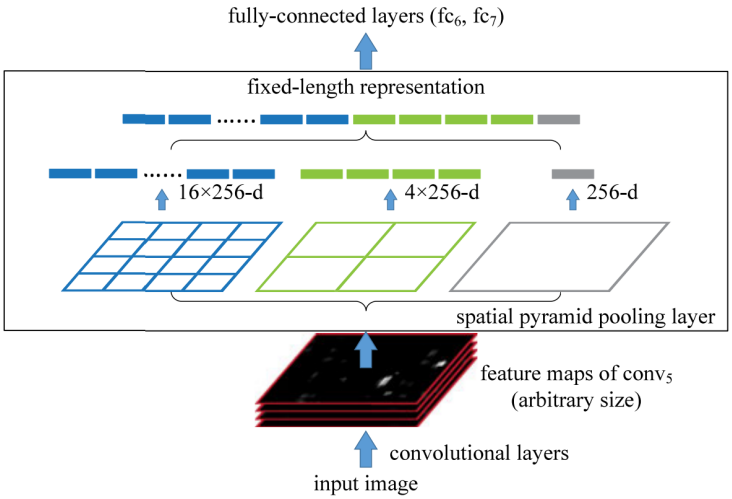

Figure 12: Pyramid pooling structure of SPPnet

During object detection training, when we import images and their target candidate regions into Fast R-CNN, both need to obtain feature categories and predicted box coordinates in the fully connected layer. Therefore, reducing the time required for the fully connected layer will greatly improve the efficiency of Fast R-CNN. Thus, truncated singular value decomposition (SVD) is introduced here. If there is a matrix W of size m x n, it can be decomposed into a matrix of size m x r, a diagonal matrix of size r x r, and a matrix of size r x n. The diagonal matrix contains important values in the top left corner and unimportant values in the bottom right corner:

$$W \approx U \sum_t V^T$$

In the fully connected layer of Fast R-CNN, W is a parameter matrix that needs to be multiplied with input features x. Thus, the equation can be changed to $Wx \approx U \sum_t V^T x$, which can be replaced with two fully connected layers: one with weight U and another with weight $\sum_t V^T$. If compression is greater, parameters are smaller, and running speed is faster. This method can improve the running speed of Fast R-CNN.

According to the article, compared with R-CNN algorithm, Fast R-CNN is 9 times faster during training and 213 times faster during testing. At the same time, its accuracy on VOC07, 2010 and 2012 datasets is also higher, making it a more efficient and accurate algorithm.

3.3.3 Faster R-CNN Detector

Faster R-CNN[37] is an algorithm that further improves the speed of Fast R-CNN. Although Fast R-CNN has greatly improved its speed and accuracy compared to R-CNN, achieving real-time detection, the real-time detection of Fast R-CNN ignores the time required to generate candidate regions. Therefore, for Fast R-CNN, the actual usage process should follow the steps shown in Figure13, which requires generating candidate regions on the image first and then operating on Fast R-CNN. However, the time required for past algorithms to generate candidate regions is actually very long, so the speed of Fast R-CNN has not been improved in this aspect. Faster R-CNN proposes a solution to the problem of long time-consuming candidate region algorithms to achieve real-time detection for the entire algorithm.

Figure 13: The structure of the Fast R-CNN convolutional neural network in practical operation

In section 3.3.1, we mentioned that R-CNN uses the Selective Search algorithm for region candidates, but it takes about 2 seconds for each image to generate region candidates. Later, a new algorithm called EdgeBoxes was created, which only takes 0.2 seconds to generate region candidates for one image, but this is still very long when there are a large number of images. Faster R-CNN uses Region Proposal Networks (RPNs) to extract features once for shared computation while satisfying the extraction required for region selection and subsequent CNN convolutional neural networks. Using this method, the candidate region generation algorithm can be reduced to 10 milliseconds per image. Figure 14 shows the structure of Faster R-CNN, which first obtains the convolutional feature map of the image through a deep convolutional network, and then imports it into the RPNs module to generate candidate regions. The subsequent operations are exactly the same as Fast R-CNN.

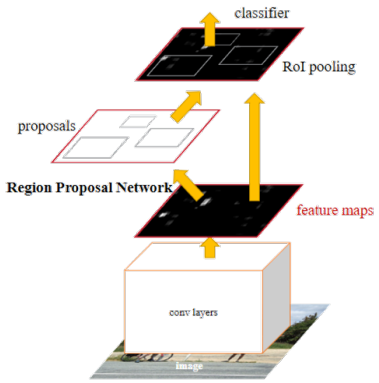

Figure 14: The structure of the Fast R-CNN convolutional neural network.

Fast R-CNN uses a time-consuming Selective Search algorithm to generate candidate regions separately, while Faster R-CNN generates candidate positions using the RPNs module on a shared convolutional feature map, so it can improve speed.

The principle of RPNs is to generate a small window with a size of n*n and continuously move this small window. Each time the window is moved, it determines whether the range defined by the window can produce a candidate region. By sliding across the entire image in this way, all candidate regions can be obtained. Each sliding window produces a 256-dimensional low-dimensional layer, which is transmitted to the classification layer and regression layer to generate categories and coordinates.

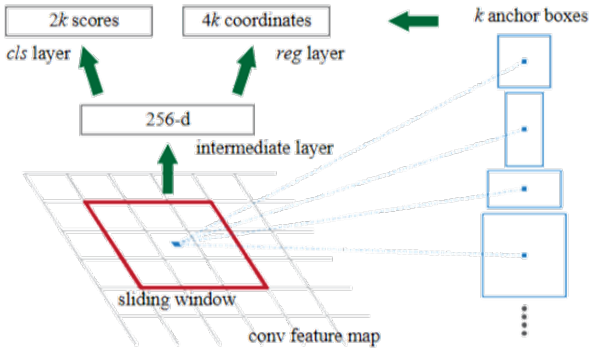

Figure 15: The structure of RPN.

3.3.4 Mask R-CNN Detector

Mask R-CNN[19] is an algorithm proposed in 2018 with a primary purpose of instance segmentation. It is currently used for various tasks such as object detection, object instance segmentation, and object keypoint detection. Compared to Faster R-CNN, Mask R-CNN adds a branch for predicting binary masks for each Region of Interest (RoI), enabling simultaneous object detection and instance segmentation. Additionally, while Faster R-CNN extracts features using coarse spatial quantization, Mask R-CNN introduces the RoIAlign method, which preserves precise spatial positions. This advancement significantly improves the performance of Mask R-CNN.

Mask R-CNN extends Faster R-CNN[37] by incorporating a mask branch, as depicted in Figure 16. The operations from the input image to the second stage are the same as in Faster R-CNN, with the difference being that Mask R-CNN passes each RoI, after RoIAlign, through separate branches for classification, regression, and mask prediction. The classification and regression branches are identical to those of Faster R-CNN.

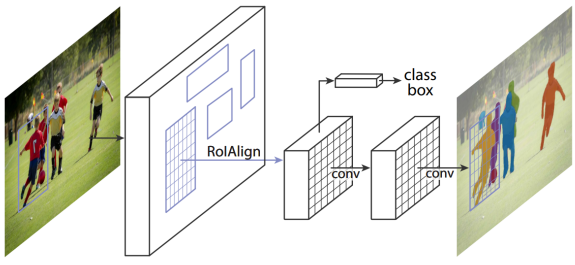

Figure 16: The structure of Mask R-CNN[19].

Faster R-CNN's pooling process suffers from certain issues, which are addressed in Mask R-CNN. Faster R-CNN employs quantization twice during the RoI pooling process, resulting in biases between the pooled features. While these biases do not affect classification, they have a significant impact on predicting pixel-accurate masks. In contrast, RoIAlign avoids quantization. It first maps RoIs from the original image to the feature map, then divides the RoIs into n×n bins and samples positions

within each bin uniformly. The values corresponding to these positions are calculated using linear interpolation and aggregated to obtain the value for each bin.

3.3.5 PointVoxel R-CNN(PV-RCNN) Detector

PointVoxel R-CNN[38], abbreviated as PV-RCNN, differs from the aforementioned 3D object detection methods by integrating convolutional neural networks (CNN) with point-based PointNet convolution, providing a better approach for understanding point cloud features. It outperforms other object detection methods on the KITTI and Waymo Open datasets. PV-RCNN combines grid-based and point-based approaches, using a multi-scale method to obtain improved proposals with high quality through voxel-based techniques and more precise local information through point-based methods.

Figure 17 illustrates the structure of PV-RCNN. The algorithm consists of two stages: voxel set abstraction for voxel representation on key points and feature extraction of key points for grid RoI. In the first stage, the entire scene is processed using voxel-based feature extraction. Simultaneously, one branch of the method applies point-based FPS sampling on the scene, utilizing only voxel features and representing them on key points. Then, the Voxel Set Abstraction (VSA) module collects features of non-empty voxels surrounding the key points and combines them. The final features are obtained through Bird's Eye View (BEV) transformation, followed by key point weight prediction. In the second stage, the previously extracted key point features are integrated into the grid points, and neighboring points within a certain radius for each grid point are identified. PointNet is employed to aggregate the features of these points into the grid points, which are then combined to form the final features.

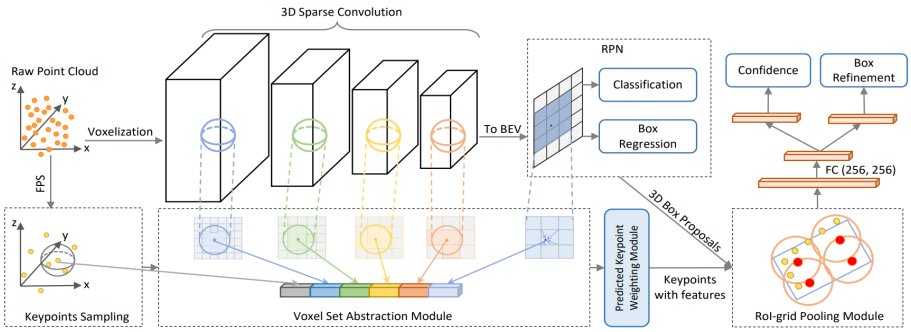

Figure 17: The structure overview of PointVoxel R-CNN[38].

4 Detectors Using Transformer

4.1 Transformer Theory

4.1.1 What Is Transformer

Transformer is a deep learning model based on a self-attention mechanism[43]. It was originally proposed by Vaswani et al in 2017 and applied to the field of natural language processing. In traditional recurrent neural networks (RNN) and convolutional neural networks (CNN), each element of input data (such as words or pixels) goes through the same transformation, while in Transformer, the transformation of each element changes based on its relationship to other elements. The Transformer model includes an encoder and a decoder for encoding an input sequence into an abstract representation and decoding that representation into an output sequence, respectively. Both the encoder and decoder are made up of multiple identical transformer layers, each containing two sub-layers of multi-head self-attention mechanisms and feed-forward neural networks. Among them, the self-attention mechanism enables the model to integrate the information of each position in the input sequence, and express different semantic information according to the attention weight between different layers and different heads. In addition, the multi-head self-attention mechanism allows the model to learn different features at different "attention heads", thus capturing the semantics of the input sequence more efficiently.

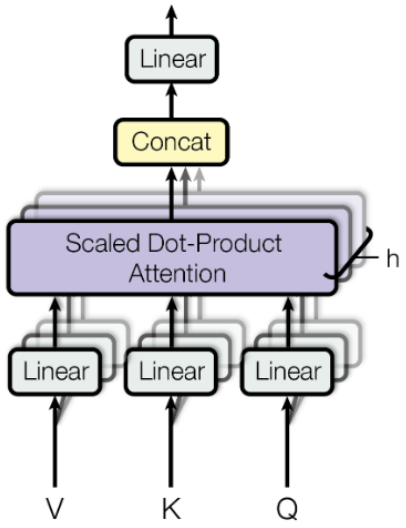

Figure 18: Multi-head Attention.

In machine learning, the input received by the neural network is often a lot of vectors of different sizes and different meanings, and there is a certain relationship between different vectors. But in the actual training, the neural network can not give full play to the relationship between these inputs, resulting in poor model training results. For this problem, self-attention mechanisms can play a good role - it is not the attention mechanism between input and output statements, but the attention mechanism between elements inside the input statement or between elements inside the output statement. In simple terms, self-attention mechanisms actually want the machine to notice correlations between different parts of the overall input. As shown in the first picture below, when we input four vectors, the self-attention system will connect these four vectors to each other, and through the corresponding matrix calculation (see the second picture below), so as to finally obtain the new four vectors and ensure that each of them has fully considered the different weights caused by different input vectors. For example, if a1 and a2 are more similar, then the final calculated b1 will be more biased towards a2.

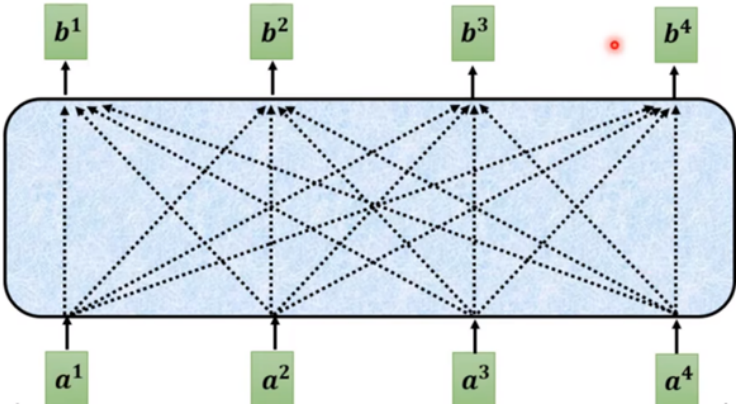

Figure 19: Each output concerned all the inputs.

With self-attention mechanisms out of the way, we'll dive into the Transformer network - the first to encode and decode using only the self-attention mechanisms just described. Compared with CNN, it can learn from the global data, not just the limited analysis of a certain window. This greatly improves the efficiency of data processing. In addition, in order to achieve the same signal multi-channel output as CNN, it uses multi-head attention to achieve optimization. Residual connection[21] and layer normalization are used to enhance the training effect of the model[1]. Figure 20 shows the

architecture of Transformer, with the encoder in the left and the decoder in the right. The encoder is composed of a stack of identical layers. Each layer has two sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, position-wise fully connected feed-forward network. Residual connection and layer normalization are used to enhance the training effect. The decoder is also composed of a stack of identical layers. In addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack. The self-attention sub-layer is also modified to ensure that the predictions for position can depend only on the known outputs at positions less than[43]. We think the encoder part is more important, the following is a detailed explanation about the encoder part.

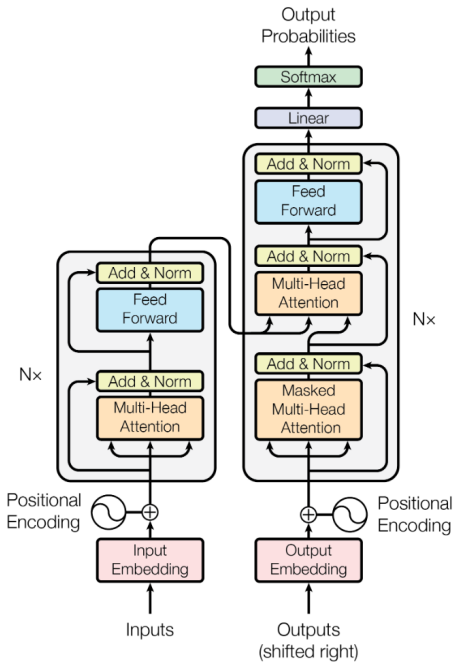

Figure 20: The Transformer model architecture.

An encoding component is a stack of encoders. A decoding component is a stack of the same numbers of decoders.

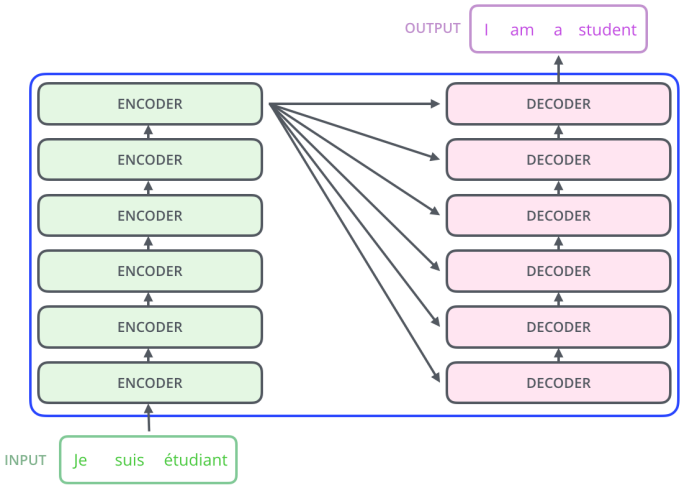

Figure 21: Stack of encoder and decoder.

Encoders have the same structure (but they have different weights). Each layer can be divided into two sub-layers.

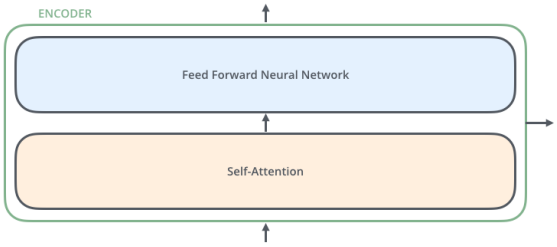

Figure 22: Stack of encoder and decoder.

The input from the encoder first flows through the self-attention layer, which helps the encoder see other words in the input sentence as it encodes a particular word. The output of the self-attention layer is then sent to the feedforward neural network. Each location is independently applied to the exact same feedforward network. Similarly, the decoder has both layers, but there is also an attention layer between them that helps the decoder focus on the relevant part of the input statement.

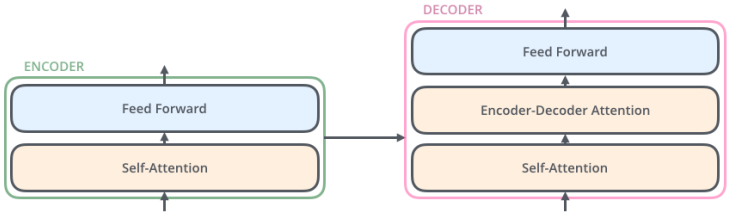

Figure 23: Sub-layers of encoder and decoder.

Typically, in an NLP application, we will first use the embedding algorithm to turn each word entered into a vector. Each word is embedded in a vector of size 512. We're going to represent these vectors with these simple boxes. Usually embedding occurs only in the lowest level of the encoder. The abstraction that holds true for all encoders is that they all receive a series of vectors, each with a size of 512 - an embedding of words in the lowest encoder, but in other encoders will be the output directly from the encoder below[[17]. The size of the vector list is a hyperparameter that we can set, which is basically the length of the longest sentence in the training data set. After embedding the words in our input sequence, each word flows separately through two layers of the encoder.

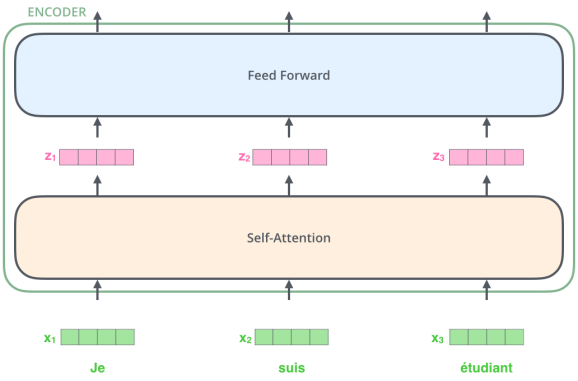

Figure 24: Embedding each words in the sentence.

As we mentioned earlier, the encoder receives a list of vectors as input. It first passes these vectors to the self-attention layer, then to the feedforward neural network, and then sends the output up to the next encoder in such a process as to process the list of vectors. The words in each position go through a self-attention process. Each of them then passes through a feedforward neural network - the exact same network, with each vector flowing independently. Transformer then uses a self-attention mechanism to encode the understanding of the relevant word into the current word.

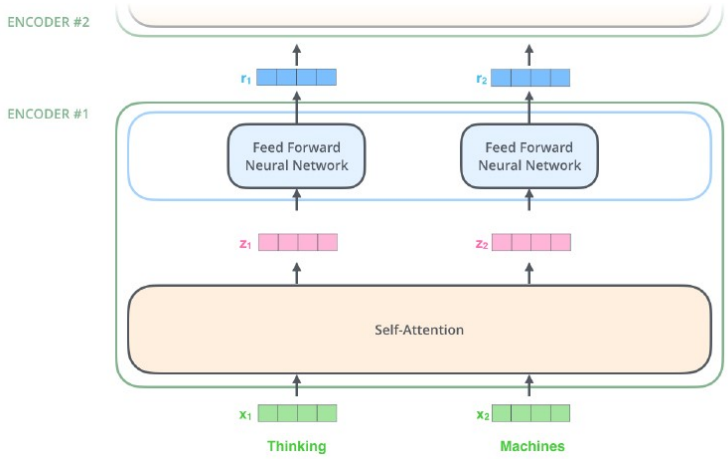

Figure 25: Words flow through encoder.

4.1.2 Why We Use Transformer

We use Transformer primarily to address the limited effectiveness and poor computational efficiency of traditional recurrent neural networks (RNNS). In traditional RNNS, the hidden state of each moment depends on the hidden state of the previous moment, which makes it difficult for the model to handle long distance dependencies, and it is not convenient for parallel computation due to the restriction of the computation order. While CNNS are able to perform well in image and speech processing, they have the same problem with sequential data. Transformer uses a self-attention mechanism and eliminates loops and convolution operations, so it can better handle long-distance dependencies in sequences and has better parallel computing performance. The self-attention mechanism can integrate the information at each position in the input sequence and calculate all positions at the same time, so as to better capture the global semantic information. In addition, Transformer uses residual joins and layer normalization during training to speed up model convergence.

Also, the 3D sparse convolutional network is a crucial component in most voxel-based detection models. Despite its advantageous efficiency, the 3D convolutional backbones cannot capture rich context information with limited receptive fields[31], which hampers the detection of 3D objects that have only a few voxels. Given the fact that the large receptive field is heavily needed in detecting 3D objects which are naturally sparse and incomplete, a new architecture should be designed to encode richer context information compared with the convolutional backbone. Recently advances[24][6][49] in 2D object classification, detection, and segmentation show that Transformer is a more effective architecture compared with convolutional neural networks, mainly because long-range relationships between pixels can be built by self-attention in the Transformer modules. Although cases in 3D is different from those in 2D, it is worthwhile to let Transformer try.

To conclusion , Transformer has the following advantages:

1. Scalability: Transformer can use more layers and more heads to enhance the expressiveness of the model, making it suitable for language models of different sizes and other sequential processing tasks.

2. The decoder is at the core: Transformer makes the decoder the most important component, making it outstanding for tasks such as machine translation and text generation.

3. Simple code implementation: The self-attention mechanisms and residuals in Transformer can be nicely encapsulated into functions, making model implementation cleaner and simpler.

Therefore, with all these advantages, Transformer has achieved great success in the field of natural language processing and has become a mainstream model for NLP tasks such as machine translation and text generation.

4.2 Detectors Using Transformer

4.2.1 Transformer-based Detectors on voxel representation

Voxel Transformer (VoTr)[31] is a Transformer-based 3D backbone that can be applied upon voxels efficiently and can serve as a better substitute for the conventional 3D convolutional backbones. Figure 26 shows its architecture. To handle the sparse characteristic of non-empty voxels, the sparse voxel module and the submanifold voxel module are built as the basic building blocks of VoTr. The submanifold voxel modules operate strictly on the non-empty voxels, to retain the original 3D geometric structure, while the sparse voxel modules can output features at the empty locations, which is more flexible and can further enlarge the non-empty voxel space. To resolve the problem that non-empty voxels are too numerous for self-attention, Local Attention and Dilated Attention are used for multi-head attention in the sparse and submanifold voxel modules. Local Attention focuses on the neighboring region to preserve detailed information. Dilated Attention obtains a large attention range with only a few attending voxels, by gradually increasing the search step.

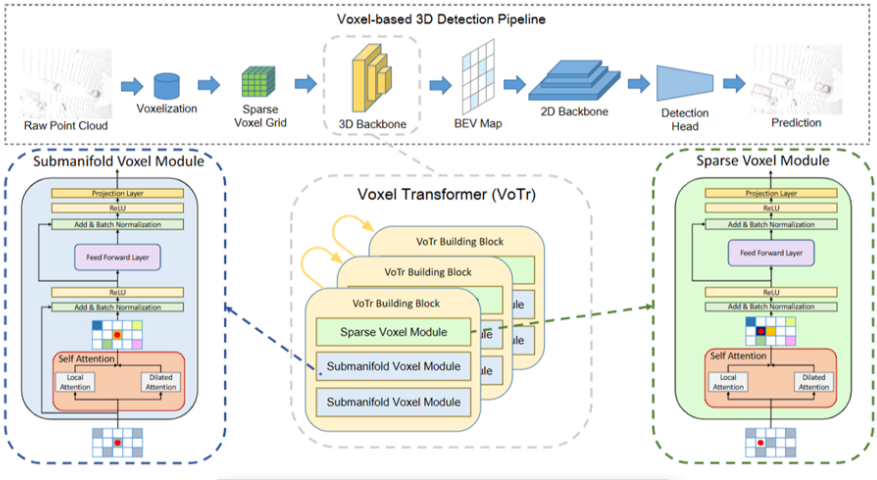

Figure 26: The architecture of VoTr.[31]

Voxel Set Transformer (VoxSeT)[18] presents a global approach to model longrange dependencies in a point cloud. It introduces a voxelbased set attention (VSA) module (see Figure 27), which consists of two cross-attentions as a replacement for the self-attention, and can process inputs of varying sizes in parallel with linear complexity. The overall architecture of VoxSeT is illustrated in Figure 28. Following the traditional transformer paradigm, the VoxSeT backbone is composed of inter-connected multilayer perception (MLP) and VSA modules. Batch norm is used as the normalization layer and each each VSA module is wrapped into a residual block for optimal gradient flow.

4.2.2 Transformer-based Detectors on BEV

Transfusion[2] uses convolution backbones to extract LiDAR BEV feature map along with an image feature map. A transformer-based decoder takes object queries as input and outputs initial bounding box predictions using the LiDAR information. Next, a spatially modulated cross attention mechanism then performs fusion between camera image features and LiDAR object queries. Figure5 shows the overall pipeline of Transfusion. The first layer produces initial 3D bounding boxes using a sparse set of object queries, initialized in a input-dependent and category-aware manner. The second layer

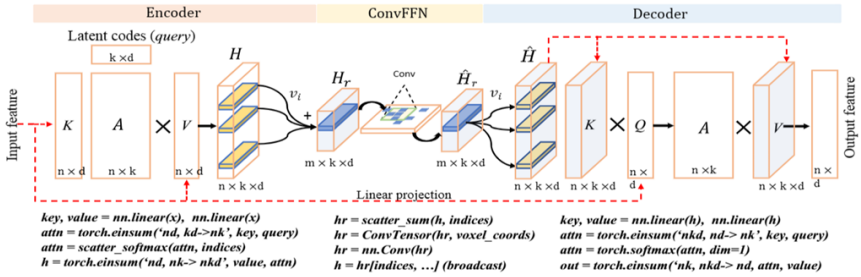

Figure 27: The matrix multiplication used in VSA.[18]

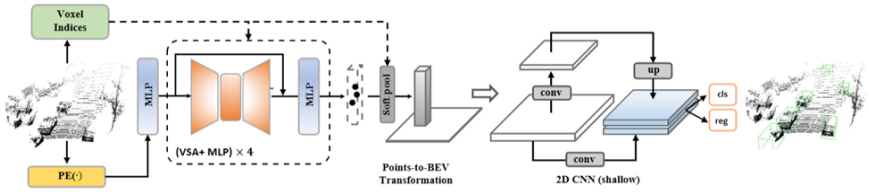

Figure 28: The overall architecture of VoxSeT.[18]

attentively associates and fuses the object queries (with initial predictions) from the first stage with the image features, producing rich texture and color cues for better detection results. A spatially modulated cross attention (SMCA) mechanism is introduced to involve a locality inductive bias and help the network better attend to the related image regions.

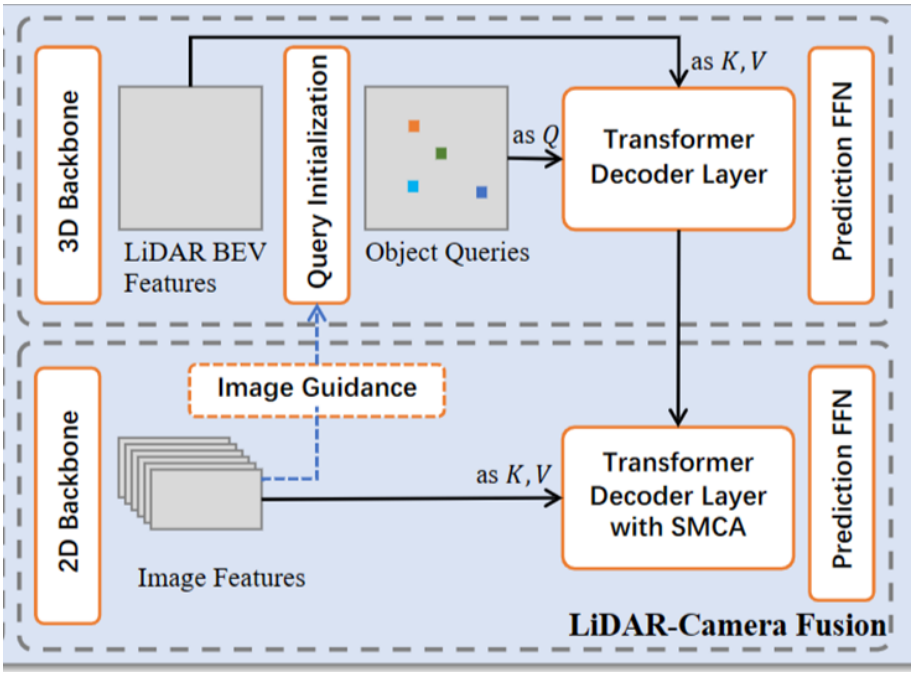

Figure 29: The pipeline of Transfusion.[2]

BoxeR[34] introduces box attention, which learns attention weights for points sampled on a grid within boxes. In 2D, it uses convolution encoder features for proposals as input and generates object queries. The object queries are then decoded into bounding boxes using instance attention. It learns

attention weights that are invariant to rotation, so another transformer is used on the bird-eye's view to generate 3D bounding boxes.

5 Between Self-Attention and Convolutional Layers

5.1 Attention vs CNN

In fact, self-attention mechanism and CNN are very similar. Self-attention mechanism does not have to be used in the field of natural language processing, it can also be used in the field of images, and its role is exactly the same as CNN after special tuning. To put it simply, CNN is local self-attention. For an image, CNN only needs local association processing, while the self-attention mechanism needs all inputs and then intercorrelation[36][8]. Therefore, when the training data is sufficient, the self-attention mechanism performs better. The picture below shows the specific performance differences between the two under different test samples and sizes. In general, Attention and CNN are two kinds of networks that similar in form but not in spirit.

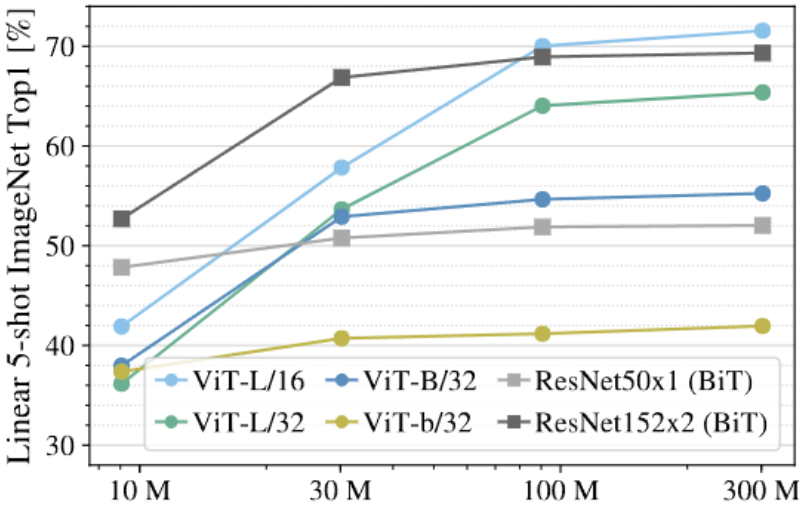

Figure 30: Self-attention vs CNN on different size of data.[24]

Attention and CNN are similar in form. Whether it is the Attention mechanism or the convolution (or full join) in CNN, at the implementation level, the data is processed in the mode of weighted summation. Weighted summation can be divided into "weighted" and "summation" two aspects, the former is to treat the data differently, while the latter is to do data fusion. So it seems that both Attention and convolution in CNN do almost exactly that. Especially in Transformer, Multi-Head Attention also has a feature fusion process in the back. In terms of operation routine, it is convolved-by-channel with CNN, and finally, it is more similar to summation along channels and feature fusion. This is also a change made by the traditional Attention system to achieve multi-channel transmission. We can even think of the convolution in the CNN as a scaled version of the Attention system that projects the full attention in the current convolution window.

Attention and CNN are different in spirit. First of all, the operation mode is different: the weight of Attention is dynamically calculated according to the input, that is, it changes with the input, which can be described as "see people talk, see ghosts talk". However, once the convolution is trained, it is dead and has nothing to do with the input. Therefore, CNN cannot show the difference between different types of input well. In contrast, Attention considers the full input, and then "highlights" under the full input, and the convolution is the window operation mode. Then there is the difference in focus: the focus of Attention is more concerned with how to decide where to project more Attention in a large frame, while convolution in CNN focuses more on obtaining another feature expression of the input signal (such as image), that is, attention is a set of mechanisms, while convolution is an operation. In addition, there is a clear concept of query (Q) and key-value data (K and V) in Transformer, which is not available in convolution.

References

[1] Jimmy Ba, JamieRyan Kiros, and GeoffreyE. Hinton. Layer normalization. *arXiv: Machine Learning*, Jul 2016.

[2] Xuyang Bai, Zeyu Hu, Xinge Zhu, Qingqiu Huang, Yilun Chen, Hongbo Fu, and Chiew-Lan Tai. Transfusion: Robust lidar-camera fusion for 3d object detection with transformers.

[3] Jorge Beltrán, Carlos Guindel, Francisco Miguel Moreno, Daniel Cruzado, Fernando Garcia, and Arturo De La Escalera. Birdnet: a 3d object detection framework from lidar information. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 3517–3523. IEEE, 2018.

[4] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152, 1992.

[5] Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11621–11631, 2020.

[6] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. *End-to-End Object Detection with Transformers*, page 213–229. Jan 2020.

[7] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-view 3d object detection network for autonomous driving. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1907–1915, 2017.

[8] Jean-Baptiste Cordonnier, Andreas Loukas, and Martin Jaggi. On the relationship between self-attention and convolutional layers. *arXiv preprint arXiv:1911.03584*, 2019.

[9] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20:273–297, 1995.

[10] Martin Engelcke, Dushyant Rao, Dominic Zeng Wang, Chi Hay Tong, and Ingmar Posner. Vote3deep: Fast object detection in 3d point clouds using efficient convolutional neural networks. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, Jul 2017.

[11] Mark Everingham, L Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes challenge 2007 (voc 2007) results (2007), 2008.

[12] Mark Everingham, Luc Van Gool, CKI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes challenge 2012 (voc2012). In *Results*, 2012.

[13] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013.

[14] Ross Girshick. Fast r-cnn. In *2015 IEEE International Conference on Computer Vision (ICCV)*, Dec 2015.

[15] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, Jun 2014.

[16] Benjamin Graham and Laurens Van der Maaten. Submanifold sparse convolutional networks. *arXiv preprint arXiv:1706.01307*, 2017.

[17] Alex Graves. Generating sequences with recurrent neural networks. *arXiv: Neural and Evolutionary Computing*, Aug 2013.

[18] Chenhang He, Ruihuang Li, Shuai Li, and Lei Zhang. Voxel set transformer: A set-to-set approach to 3d object detection from point clouds.

[19] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, page 1904–1916, Sep 2015.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[22] Xinyu Huang, Xinjing Cheng, Qichuan Geng, Binbin Cao, Dingfu Zhou, Peng Wang, Yuanqing Lin, and Ruigang Yang. The apolloscape dataset for autonomous driving. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 954–960, 2018.

[23] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

[24] Alexander Kolesnikov, Alexey Dosovitskiy, Dirk Weissenborn, Georg Heigold, Jakob Uszkoreit, Lucas Beyer, Matthias Minderer, Mostafa Dehghani, Neil Houlsby, Sylvain Gelly, Thomas Unterthiner, and Xiaohua Zhai. An image is worth 16x16 words: Transformers for image recognition at scale. 2021.

[25] Jason Ku, Melissa Mozifian, Jungwook Lee, Ali Harakeh, and Steven L Waslander. Joint 3d proposal generation and object detection from view aggregation. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–8. IEEE, 2018.

[26] Alex H. Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jan 2020.

[27] Bo Li. 3d fully convolutional network for vehicle detection in point cloud. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Dec 2017.

[28] Bo Li, Tianlei Zhang, and Tian Xia. Vehicle detection from 3d lidar using fully convolutional network. In *Robotics: Science and Systems XII*, Jun 2016.

[29] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.

[30] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*, pages 21–37. Springer, 2016.

[31] Jiageng Mao, Yujing Xue, Minzhe Niu, Haoyue Bai, Jiashi Feng, Xiaodan Liang, Hang Xu, and Chunjing Xu. Voxel transformer for 3d object detection. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, Mar 2022.

[32] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[33] Alexander Neubeck and Luc Van Gool. Efficient non-maximum suppression. In *18th international conference on pattern recognition (ICPR'06)*, volume 3, pages 850–855. IEEE, 2006.

[34] Duy-Kien Nguyen, Jihong Ju, Olaf Booij, Martin R. Oswald, and Cees G. M. Snoek. Boxer: Box-attention for 2d and 3d transformers. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2022.

[35] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[36] Maithra Raghu, Thomas Unterthiner, Simon Kornblith, Chiyuan Zhang, and Alexey Dosovitskiy. Do vision transformers see like convolutional neural networks? *Advances in Neural Information Processing Systems*, 34:12116–12128, 2021.

[37] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

[38] Shaoshuai Shi, Chaoxu Guo, Li Jiang, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng Li. Pv-rcnn: Point-voxel feature set abstraction for 3d object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10529–10538, 2020.

[39] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[40] Shuran Song and Jianxiong Xiao. Deep sliding shapes for amodal 3d object detection in rgb-d images. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 808–816, 2016.

[41] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, et al. Scalability in perception for autonomous driving: Waymo open dataset. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2446–2454, 2020.

[42] Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 1999.

[43] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[44] Yan Yan, Yuxing Mao, and Bo Li. Second: Sparsely embedded convolutional detection. *Sensors*, page 3337, Oct 2018.

[45] Bin Yang, Ming Liang, and Raquel Urtasun. Hdnet: Exploiting hd maps for 3d object detection. *Conference on Robot Learning*, Oct 2018.

[46] Bin Yang, Wenjie Luo, and Raquel Urtasun. Pixor: Real-time 3d object detection from point clouds. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Dec 2018.

[47] Maosheng Ye, Shuangjie Xu, and Tongyi Cao. Hvnet: Hybrid voxel network for lidar based 3d object detection. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Aug 2020.

[48] Dominic Zeng Wang and Ingmar Posner. Voting for voting in online point cloud object detection. In *Robotics: Science and Systems XI*, Jan 2016.

[49] Sixiao Zheng, Jiachen Lu, Hengshuang Zhao, Xiatian Zhu, Zekun Luo, Yabiao Wang, Yanwei Fu, Jianfeng Feng, Tao Xiang, Philip H.S. Torr, and Li Zhang. Rethinking semantic segmentation from a sequence-to-sequence perspective with transformers. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2021.

[50] Yin Zhou, Pei Sun, Yu Zhang, Dragomir Anguelov, Jiyang Gao, Tom Ouyang, JamesC.Y. Guo, Jiquan Ngiam, and VijayK. Vasudevan. End-to-end multi-view fusion for 3d object detection in lidar point clouds. *Conference on Robot Learning*, Jan 2019.

[51] Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Dec 2018.