Current Approaches and Future Directions for Point Cloud Object Detection in Intelligent Agents

Kyle Vedder

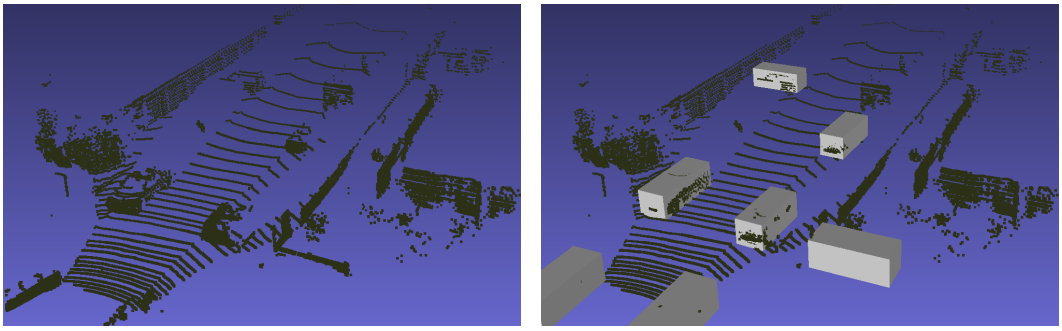
April 14, 2021

Abstract

The explosion of commercial interest in self-driving cars has led to an outpouring of new methods for point cloud-based object detection in the academic literature. Much of this literature focuses on improving object detection accuracy metrics for static datasets taken from self-driving car domains, but real world constraints such as runtime and compute are often only discussed in passing, resulting in hurdles for practitioners when attempting to evaluate and deploy these approaches to real world intelligent agents such as self-driving cars or service robots. This work surveys various current approaches, critiques this work with an eye on deployment considerations for intelligent agents, and outlines future directions of research that focus on these considerations.

1 Introduction

A point cloud is an unordered collection of 3D points that represent a scene via their location on the surface of objects, e.g. Figure 1. Point clouds can convey rich 3D information about a scene, such as object scale and relative positions, which would be difficult or impossible to extract from other common scene representations such as images. This rich 3D information is important to intelligent agents that want to understand the real world and critical to embodied intelligent agents that need to ensure safety. In point clouds, object detection typically takes the form of bounding box regression, i.e. fitting a rotated rectangle to contain



(a) Raw point cloud (b) Point cloud with regressed cars

Figure 1: Example point cloud from the KITTI Vision Benchmark Suite [1]

a discrete object, along with a classification label for that object. As an example, Figure 1b depicts gray bounding boxes regressed onto several cars in the point cloud.

In the last few years, self-driving cars have emerged as the most well-known class of embodied intelligent agents, and point cloud-based object detection has been a cornerstone of their ability to understand and navigate the real world. Due to the tremendous interest in developing robust self-driving cars, significant advancements have been made in every part of the point cloud processing pipeline, from the processing algorithms to the computational hardware to the sensors themselves, allowing self-driving cars to be smarter and safer. In this work we do a survey of the academic literature on various current approaches to point cloud-based object detection by constructing a taxonomy of various classes of approaches and considering representative work. We then critique this work with an eye on the constraints that come with intelligent agents deployed in the real world, such as execution time, which must be kept low in order for the agent to safely react to the highly dynamical environments, and compute constraints, which are tight due to the necessary use of on-board computation. Finally, we outline several future directions of research to solve many of these challenges.

As self-driving cars and service robots are embodied agents deployed in highly dynamic environments, this work will only consider object detection approaches for single egocentric observations. There is prior art on processing point clouds fused together from multiple observations [2, 3], but these are necessarily constructed from asynchronous observations; older observations quickly become out-of-date due to the dynamic nature of these environments, severely curtailing their usefulness.

The rest of this work proceeds as follows: First, we outline how methods are evaluated and the datasets they are commonly evaluated on (Section 2). Second, we present a taxonomy of common classes of approaches and discuss several representative methods, considering the value proposition and open questions each method presents to practitioners (Section 3). Third, we discuss issues with the literature from a practitioner's perspective and propose new directions of research in order to meet these needs for real-world intelligent agents (Section 4)

2 Standard Benchmarks for Object Detection

As stated in Section 1, the objective of object detection in point clouds is to classify and regress bounding boxes onto all relevant objects in the scene. All current approaches treat this as a supervised learning problem, with datasets supplying a ground truth bounding box and a class label for each relevant object in the scene. One of the first major datasets for self-driving cars to do this was KITTI [1], released in 2012. This dataset provides approximately 15,000 hand labeled point clouds from a suburban and urban setting, split evenly into a train and test set, collected via a Velodyne HDL-64e LIDAR[1]. To evaluate performance, KITTI uses the Intersection over Union (IoU) measurement for accuracy used by the image object detection dataset PASCAL VOC [4], shown in Equation 1,

$$IoU = \frac{\text{volume}\left(B_{\text{proposed}} \cap B_{\text{truth}}\right)}{\text{volume}\left(B_{\text{proposed}} \cup B_{\text{truth}}\right)} \tag{1}$$

adapted to 3D bounding box volume instead of 2D image box area, along with the same algorithm for deduplicating predictions and matching them to ground truth boxes. As KITTI

[1] `https://velodynelidar.com/products/hdl-64e/`

is one of the oldest datasets, it is by far the most common benchmark in the literature, with their public leaderboard featuring over 100 submissions from published work[2]. Unfortunately, due to the relatively small size of the dataset, modern methods must engage in significant data augmentation in order to prevent overfitting, e.g. splicing cars from other scenes into the dataset. Additionally, the LIDAR sensor used to generate the point clouds produces low point density at range, making evaluation difficult for far away objects, critical for things like highway driving.

Other datasets have been released in the past two years which better serve these needs, but are far less heavily used in the literature. The Argoverse dataset [5], released in 2019, features roughly 11,000 objects tracked for 15 to 30 second continuous intervals on a variety of suburban and urban environments. The point clouds were collected with two LIDARs on top of the car as well as a forward facing stereo camera pair in order to provide significantly more range and point density in the point clouds. The Waymo Open dataset [6], released in 2019, features roughly 10 million objects tracked for 20 second continuous intervals over a variety of suburban and urban environments, across weather conditions. The point clouds were collected with one custom long-range LIDAR and four custom short range directional LIDAR in order to provide high density coverage for close objects and coverage at range. The NuScenes dataset [7], released in 2019, features roughly 1 million objects tracked for 20 second continuous intervals over a variety of suburban and urban environments. The point clouds were collected with one LIDAR and five RADAR. All three datasets feature the same KITTI IoU accuracy scoring protocol, and all three sponsoring companies have launched public competitions for their datasets, providing new evaluation platforms for literature.

However, despite new datasets alleviating size and sensor issues with KITTI, none of these dataset evaluation protocols consider the constraints practitioners must contend with when selecting a method for a real system. For example, methods are primarily evaluated on their accuracy, thus encouraging methods to drastically increase their runtime or compute requirements in order to squeeze a few percentage points of additional accuracy out of their method, a trade-off that is rarely reasonable on real systems as detections are time sensitive and resources constrained. In Section 4, we discuss ways of encoding some of these requirements into new evaluation protocols and propose directions of research that consider this type of deployment constraint.

3 Taxonomy of Point Cloud-based Object Detection Approaches

Broadly speaking, current approaches to point cloud-based object detection reason about the scene in (at least) one of three ways: 1) chunking the scene into 3D Voxels and processing each voxel (Section 3.1), 2) processing the scene from a top-down Birds-Eye View (Section 3.2), and 3) processing the scene from an egocentric view via a Range Image (Section 3.3). For each detection approach, we review representative state-of-the-art methods and discuss open questions from these methods.

3.1 3D Voxels

In 2D observations, images are comprised of pixels arranged in a regular grid-like structure that contain grayscale or color information about the scene. This regular structure allows

[2] `http://www.cvlibs.net/datasets/kitti/eval_object.php?obj_benchmark=3d`

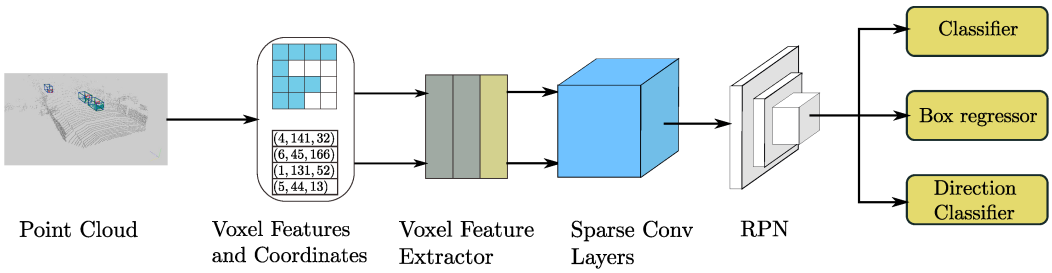


Figure 2: Architecture diagram from SECOND [10] paper.

2D convolutional filters to be efficiently applied to local clusters of pixels in order to extract local information about the scene. In 3D, an analogous structure is called a *voxel*, a 3D box in a regular grid-like structure that contains information about that region of the 3D scene. Similarly, this regular structure allows 3D convolutional filters to be efficiently applied to local clusters of voxels in order to extract local information about the scene. Thus, a straight forward approach to processing 3D scene data is to vectorize the points inside each voxel using a PointNet-like [8] vectorizer (see Appendix A for details), perform 3D convolutions over the voxels to extract local features, and then pass this into a backbone to perform bounding box regression and classification; in essence, this is how VoxelNet [9] operates.

Unfortunately, this straight-forward approach suffers from significant computational overhead. In 2D images the number of pixels scales with the area of the image (quadratic), whereas in 3D point clouds the number of voxels scales with the volume of the scene (cubic). Thus, in order to produce voxels small enough to capture fine details required for classes like pedestrians or cyclists, a very large number of voxels must be used and thus convolved. As a result, these 3D convolutions dominate VoxelNet's runtime.

However, despite there being over 100,000 points per laser scan in the KITTI dataset, typically well over 90% of VoxelNet's voxels are empty. This is the result of the fundamental shape of real-world environments; most voxels do not correspond to the observable surface on an object. This motivates Sparsely Embedded Convolutional Detection (SECOND) [10], a 3D convolutional approach which performs bookkeeping in order to avoid convolving the many empty voxels, inspired by the approach behind image based SparseConvNet [11, 12].

An overview of SECOND's architecture, which is very similar to VoxelNet, can be seen in Figure 2. Both perform the same voxelation of the point cloud, both employ the same technique for downsampling overfull voxels and converting them to vectors, both perform 3D convolutions (though SECOND uses its sparse approach), and both pass the result to a Faster R-CNN style Region Proposal Network (RPN) [13] to regress and classify bounding boxes (though SECOND modifies its RPN).

An example of SECOND's sparse convolutional approach for a 2D 3×3 convolutional filter is illustrated in Figure 3. First, a Rule matrix is preconstructed providing an offset mapping from input coordinates to output coordinates as per the size and stride of the convolutional filter. Then, at runtime, the non-zero input features are gathered, along their coordinates, and can be directly converted to the output result via general matrix multiplication (GEMM), and then scattered back into the sparse output space using the coordinates transformed by their multiplication with the Rule matrix.

SECOND's RPN is modified in two major ways. First, the RPN regresses on a range from $-90°$ to $90°$ with a second binary classification target for direction; this approach

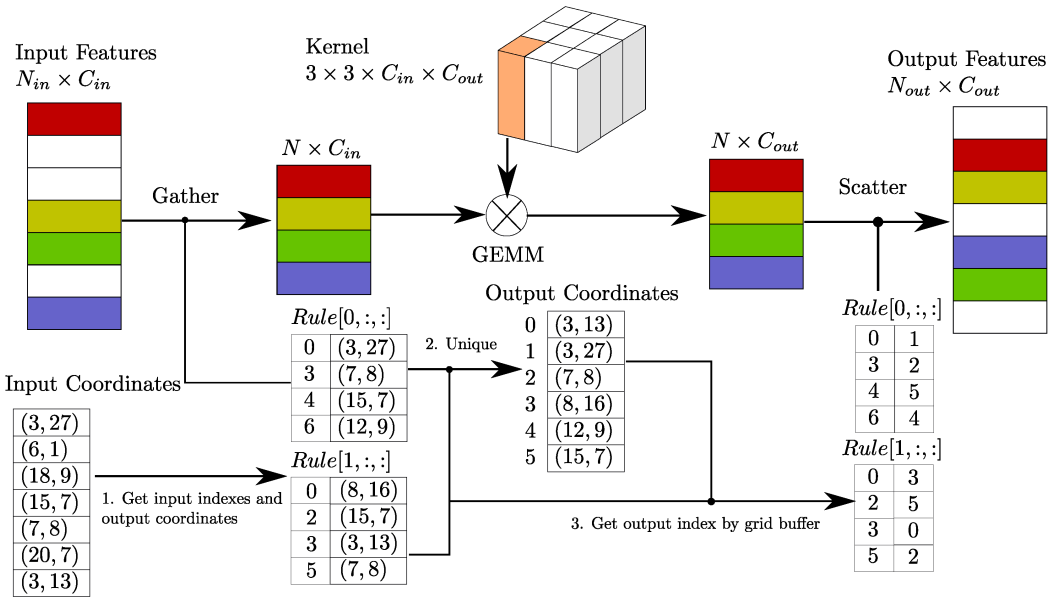


Figure 3: Sparse convolution example from SECOND [10] paper depicting a 2D 3×3 convolutional filter for visualization purposes.

separates orientation from angle, allowing the RPN to independently receive feedback on both as part of the loss. Second, the RPN does not attempt to regress box length, width, or height; instead, it only regresses the center, and the average size box for the predicted class is inserted, removing three degrees of freedom from the regession output. This approach works because there is little variance of box shape within a class on the KITTI dataset.

SECOND is presented in two variants: a large variant with high quality results and a small variant which produces slightly lower quality results but takes less than half the time to train and twice as fast inference speed. The full details of the model architectures, data augmentation, optimizer settings, the GPUs used to train them are provided in the paper, and a public implementation from the authors is available online[3].

The evaluation section compares SECOND to prior art on the KITTI dataset for 3D object detection. While running inference in 0.05 seconds, fast enough for 20Hz inference, SECOND outperforms prior art on car detection, including VoxelNet, presumably due to its improvements to the RPN and data augmentation, and is competitive with prior art on person and bicyclist detection. The authors briefly discuss failure modes: false negatives on far away, highly sparse observations of cars due to very few points on cars, and higher rates of false positives and false negatives in pedestrians due to their smaller size relative to the size of the voxels. Additional evaluation is performed to demonstrate the performance improvement from changing the RPN's angle regression approach relative to prior art, and training with their data augmentation approach demonstrates a significant improvement over the same architecture without their data augmentation approach.

From a practitioner's perspective, this work is well motivated and provides significant detail on the implementation of the sparse convolutional approach. The changes to the RPN

[3] `https://github.com/traveller59/second.pytorch`

in order to improve its angular regression over VoxelNet are well explained and the authors provide an intuition as to why the changes improve performance. The significantly smaller network with only slightly degraded performance hints at the possibility that the presented networks can be further scaled down while retaining similar performance properties, allowing for reasonable inference time on compute constrained systems. However, this work also leaves unanswered several questions important to practitioners:

1. SECOND uses the same voxel configuration as VoxelNet, and neither explore other configurations. How does SECOND perform when voxels are made smaller or larger? Voxel size is cited as a limiting factor in the accuracy of pedestrian detection, but decreasing their size would not only significantly increase inference time, it could potentially have detrimental effects to detection quality due to voxels containing only a small number of points.

2. Unlike VoxelNet, SECOND does not regress the size of the bounding box as part of its RPN; it predicts the center of the box along with its rotation, and then inserts the average sized box associated with that class at that position and orientation. This approach only works because each class in the KITTI dataset is roughly the same shape. How much of SECOND's performance improvement over VoxelNet on the KITTI dataset can be attributed to this dataset dependent simplification?

3. The paper presents the large and small versions of SECOND, as well as minor changes to the RPN for cars versus pedestrians; however, it provides no intuition as to why the authors made the changes they did, or why the changes resulted in only minor quality degradation. What motivated these changes, and what other reasonable configurations options exist?

4. Related to Question 3, the paper provides no data on how much each network component contributes to the total inference time. What components of the network can be changed in order to most directly reduce inference time?

3.2 Birds-Eye View

As discussed in Section 3.1, the number of voxels scales with scene volume; this cubic scaling directly contributes to performance issues faced by approaches that operate on 3D voxels. An alternative approach is to reason about the point cloud from a top-down, birds-eye view; this view scales with horizontal scene area rather than volume and allows methods to reason about the scene via 2D features and thus leverage efficient 2D detection networks for their detection head.

A straight forward approach to processing 3D scene data is to discretize the scene into a grid of *pillars* that contain all data along the vertical axis, vectorize the points in each pillar into embeddings using a PointNet-like [8] learned feature extractor, thus forming a *pseudoimage* with the dimensions of the pillar grid and channels equal to the size of the vector embedding, and then use a traditional 2D convolution-based image backend to classify and regress bounding boxes. In essence, this is how PointPillars [14] operates; its full architecture depicting this pipeline is shown in Figure 4.

Like with SECOND, PointPillars leverages point sparsity in order to reduce its memory footprint and computational cost. The vast majority of pillars are empty due to point cloud sparsity (an average of 97% of the pillars were empty for their KITTI experiments), so PointPillars gathers the set of non-empty pillars into a fixed size dense tensor with a

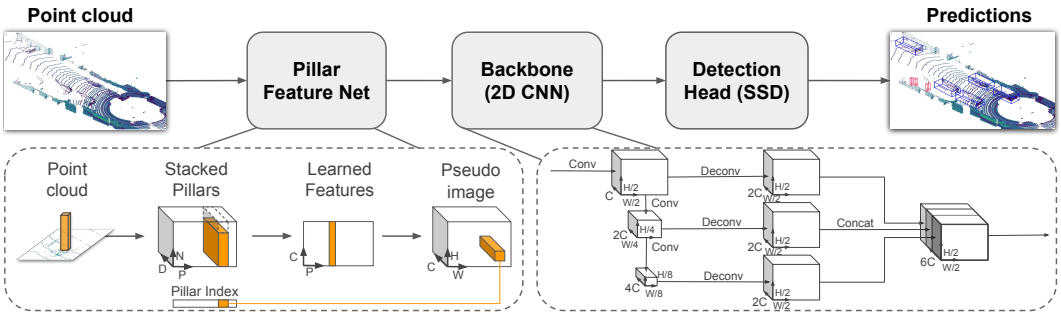


Figure 4: Architecture diagram from PointPillars [14] paper.

fixed number of points per pillar; excess pillars are discarded and pillar points are padded or subsampled to hit the per-pillar thresholds. This dense tensor representation enables its PointNet-like feature extractor to efficiently vectorize each pillar (see Appendix A for details) and scatter the resulting vector into a full pseudoimage to then be processed by the CNN backbone.

The CNN backbone is similar in principle to the convolutional backbone of VoxelNet and SECOND; it constructs a feature pyramid via strided convolutions, concatenating transformed outputs at each level to pass into the final regression head. Unlike VoxelNet or SECOND, which employ a Faster R-CNN style RPN [13], PointPillars uses a Single Shot Detector (SSD) [15] regression head, presumably due to its superior performance on image datasets. Bounding boxes are regressed as 2D boxes in the birds-eye view, with height and elevation regressed as secondary targets.

The evaluation section compares PointPillars to prior art on the KITTI dataset. In its default configuration, PointPillars performs inference in 0.0162 seconds, fast enough for inference at 60Hz, while almost universally outperforming all prior art in 3D object detection across all three object classes and producing the highest mean AP on the Birds-Eye View and 3D detection benchmarks. The authors provide full details on the network configuration, the loss function, the optimizer configuration, and the pillar configurations, which differ between the car and pedestrian & bicyclist detection problems due to different max ranges. Full details of the data augmentation regime are presented, along with their genesis relative to SECOND's approach. Runtime for the default configuration is analyzed in detail; runtimes for each section of the network are presented, along with subsections for each major part detailing the intuition behind the design decisions leading to the runtime results. Two ablative studies are performed. First, pillar size is repeatedly increased, showing significantly faster inference for only minor loss in mAP (tradeoff curve is shown in Figure 5). Second, PointPillars' vectorizer is compared against VoxelNet's vectorizer as well as other approaches by implementing them all inside SECOND and trained on KITTI using PointPillars' data augmentation techniques; PointPillars' vectorizer produces the same or better results compared to other encoders despite being significantly smaller. The authors note that they achieved better performance using SECOND with its default encoder than presented in the official paper; this most likely caused by their improved data augmentation approaches, indicating the efficacy of their data augmentation approaches.

From a practitioner's perspective, this work provides an effective and well-validated engineering solution along with extensive intuition and characterization of its architecture. The full network architecture, loss, and optimizer configurations are listed, and the full

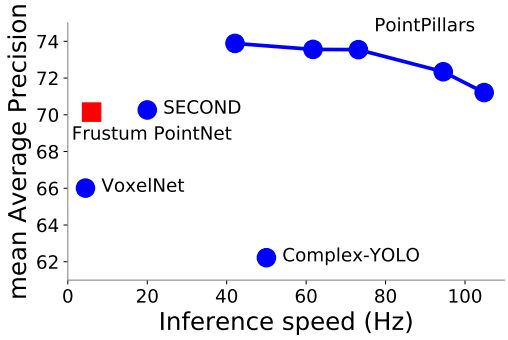


Figure 5: Inference speed vs BEV mAP for different pillar sizes of PointPillars from [14].

data augmentation is clear. The exploration of the impact of pillar size versus inference speed and quality clearly characterize the tradeoff curve, providing an immediate direction for finetuning. The discussion of component runtimes also provides additional directions of runtime improvement, both from an architecture perspective (e.g. smaller and faster alternatives to SSD can be explored) and an implementation perspective (e.g. LIDAR points can be streamed directly to the GPU, skipping the copy step which takes almost 20% of the total inference time). A public implementation is available online from the authors[4]. Despite the robust exploration and experimentation, several practical questions remain:

1. As per the inference runtime breakdown in Section 6, the largest single processing chunk was the CNN backbone and processing heads. As mentioned in the ablation studies, changing the resolution of the pillars changed the size of the pseudoimage and thus the size of the CNN backbone, contributing to faster runtimes, but there was no discussion regarding why the SSD regression head was chosen or possible architectural changes in this backend to change performance or inference speed. Why was SSD chosen, can it be directly ablated, and what properties are needed to make other detectors good detection heads?

2. The experimentation confirmed that, despite being significantly smaller, PointPillars' vectorizer provided equal performance to VoxelNet's vectorizer. What further changes can be made to the vectorizer that allow for a runtime versus quality trade-off? Can the vectorizer be shrunk further?

3.3 Range Images

Section 3.1 and Section 3.2 both discuss various ways of partitioning the scene into a regular grid structure and processing each region. This regular structure means an inherent lack of scale ambiguity, as each voxel or pillar covers the same size area, as well as better handling of occlusions, as the negative space of the object is natively encoded in the representation, but this structure also leads to high sparsity, as most voxels or pillars remain empty. Depth images, i.e. an image from the perspective of the agent with depth measurements contained in each pixel (e.g. Figure 6), have the opposite properties. Depth images are dense: every pixel in the depth image corresponds to a reading from the sensor; however, the depth of

[4] `https://github.com/nutonomy/second.pytorch`

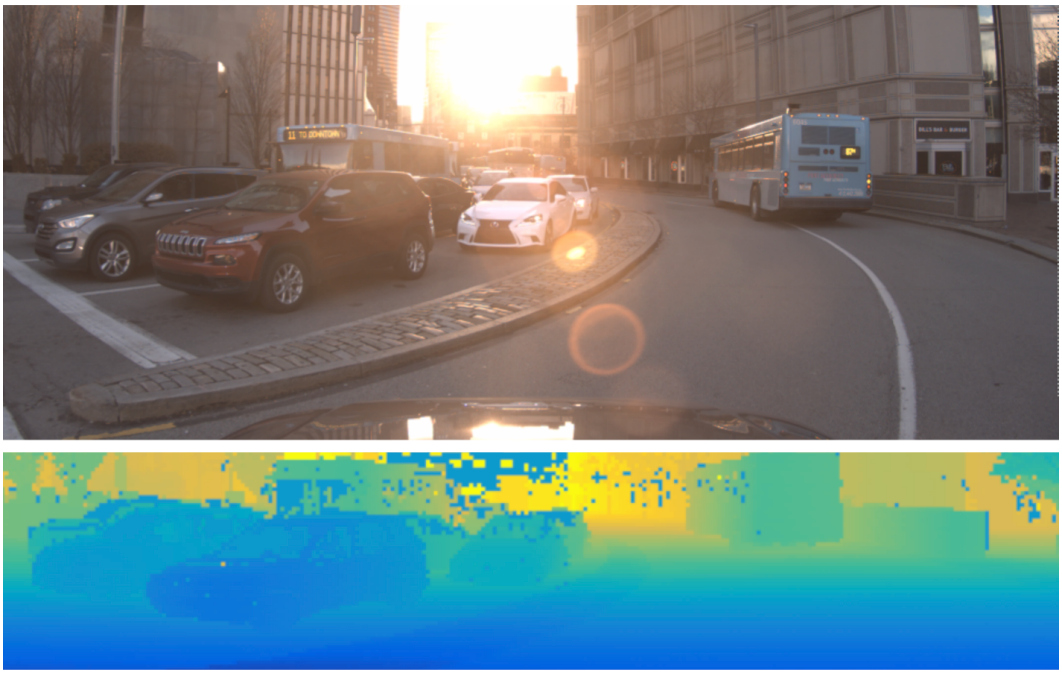


Figure 6: Example RGB image and associated Depth Image from LaserNet [16] paper.

each pixel must be taken into account when reasoning about local neighborhood and care must be taken in order to handle occlusions.

The general architecture for LaserNet is shown in Figure 7. Given a point cloud, a five-channel depth image containing range, height, reading azimuth, intensity, and an return flag (indicating if a reading produced a return) is passed into a CNN which estimates class probability, full object bounding box, and box corner variance for every pixel. These boxes are then merged using a custom clustering approach based on Non-Maximal Suppression that reasons about the predicted uncertainty of each estimate.

The CNN's architecture is shown in Figure 8. The network is modeled using residual networks [17] as modules stacked in a feature pyramid network, akin to those used in classical

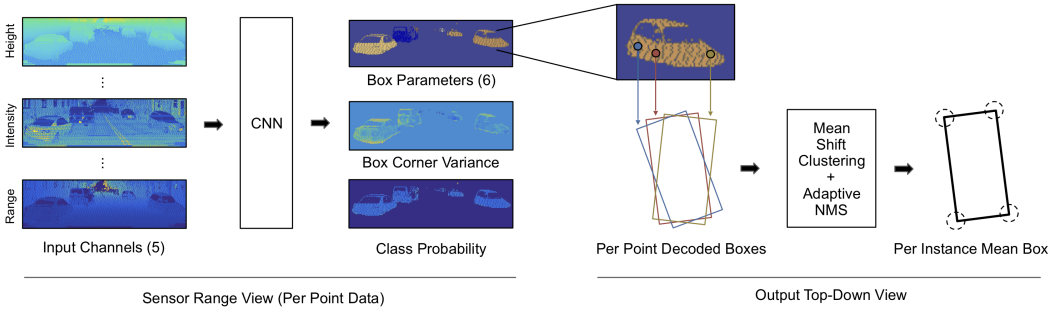
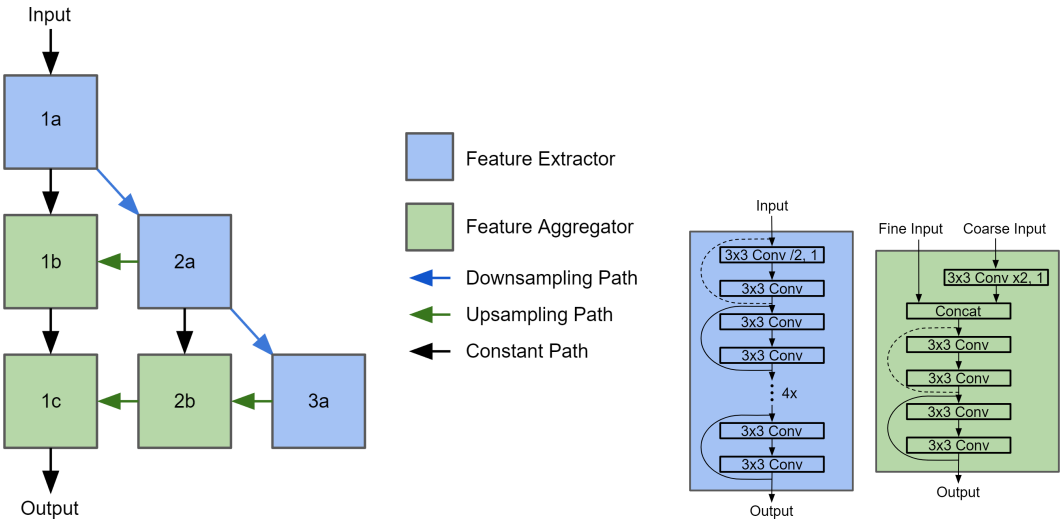


Figure 7: Architecture diagram from LaserNet [16] paper.



(a) CNN Module Composition (b) CNN Modules

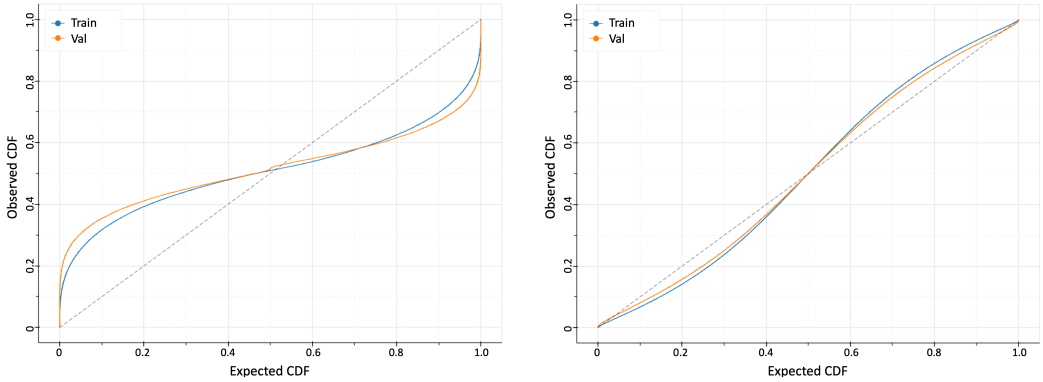
Figure 8: CNN Architecture from LaserNet [16] paper.

feature extractors [18, 19, 20], in order to handle scale changes due to object distance. A final 1×1 convolutional layer is then applied to directly emit the box predictions for each pixel.

The box estimates are then merged together using a mean shift clustering [21] approach. Every bounding box is assumed to be touching the ground plane, reducing the merge problem to 2D, where a custom 2D specific approximate version of mean shift clustering is run over the center estimate of every predicted bounding box in order to extract the most likely box given the discrete samples. This custom algorithm discritizes the space along the ground plane and repeatedly performs regression box merging in means fall within the same box. This approach can be implemented on the GPU, and in the experimental results was run for a small, fixed number of iterations with a relatively large grid size (0.5 meters).

In most models, non-maximal suppression is used to de-duplicate predicitons by removing bounding box predictions that have a high IoU with another predicted box but a lower class confidence. However, this model provides estimates of box variance, not simply class confidence, and so the non-maximal suppression algorithm is modified to take both of these features into account when deciding which bounding boxes to suppress.

The authors compare LaserNet to prior art on the KITTI dataset as a small scale experiment and the ATG4D dataset, a non-public dataset at Uber Advanced Technologies Group (Uber ATG), as a large-scale experiment. Due to the non-public nature of ATG4D, only other Uber ATG works can compare against on this dataset, and Uber ATG has not published results for non-ATG developed methods in the literature on this dataset. LaserNet performs poorly on the KITTI dataset, performing approximately 10% worse than the best performing baselines. However, the authors argue that, even with data augmentation, the KITTI dataset is too small to fully train their large network, hence their reliance on ATG4D. On ATG4D, LaserNet slightly outperforms other models overall, but slightly underperforms on cars further than 50 meters away and bikes further than 30 meters away. In these situations, the dominant method uses LIDAR + RGB images which provides additional useful



(a) KITTI CDF (b) ATG4D CDF

Figure 9: CDFs of predicted versus actual occurrences of object for LaserNet [16].

signal at these ranges due to the relatively low resolution nature of depth readings. Despite being a large model, the evaluation time of LaserNet is approximately 0.03 seconds, making it faster than SECOND for inference speed, as it is almost entirely convolutional.

In order to validate some of the unusual design decisions in the model, the authors claim to have tested their model without these features. First, when distribution prediction is replaced with mean prediction, their problem formulation collapses down to simple box merging and NMS, which the authors claim produce worse results. Second, they claim using raw sensor data with non-uniform azimuth changes, instead of reprojecting the input depth to have uniform azimuth changes, results in higher accuracy. Third, they claim their box merging approach results in better distribution estimation than any individual distribution represents. Forth, they claim their custom NMS approach allows LaserNet to maintain its probabilistic estimates of bounding boxes, leading to better accuracy. However, it is not clear what tests were run to lead to these conclusions, as they are only discussed in passing.

In order to validate the quality of the probabilistic reasoning performed by the model, a CDF of predicted versus actual occurrences is plotted for KITTI and ARG4D, shown in Figure 9. Figure 9a shows that, on the KITTI dataset, LaserNet aggressively tends towards predicting around 50%, causing it to be significantly overconfident in low probability cases and significantly underconfident in high probability cases, whereas Figure 9b shows that, on the ATG dataset, LaserNet is well calibrated, with only slight underconfidence on low probability cases and slight overconfidence in high probability cases. The authors argue this disparity is caused by the dataset size difference; the much larger size of the ATG4D dataset allows LaserNet to fully calibrate itself, leading to its much better performance on the KITTI dataset.

From a practitioner's perspective, this work provides a compelling case for predicting probability distributions, rather than simple means, as a way to develop a more robust detector. This finding is tempered by the requirement of needing large labeled datasets and significantly more computational resources to train the model. Additionally, this work provides a compelling case for range image based piplines. Despite being a deep model, the compactness of the input due to the density of range images allows for the model to be competitive with 3D voxel methods such as SECOND in terms of runtime. However, this work suffers from a number of problems. No public implementation of LaserNet exists,

the primary dataset used in the experiments is not public, and the other approaches compared against on this dataset lack a public implementation. As a result, this work leaves unanswered several questions important to practitioners:

1. KITTI was deemed too small to provide robust probabilistic estimates for bounding boxes, leading to poor results, whereas ATG4D was deemed sufficiently large to provide well-calibrated predictions, leading to good results. How large of a dataset is needed in order to provide well-calibrated results? What other properties, such as positional diversity, does this dataset need to have in order to ensure well-calibrated results?

2. The CNN architecture is a large feature pyramid comprised of three layers of fairly large modules, but there was no discussion regarding changing the size or configuration of these modules. How does removing a level of the feature pyramid impact prediction quality, runtime, and calibration? How does removing layers from these modules impact prediction quality, runtime, and calibration?

3. The authors claim that LaserNet performs better on ATG4D due to the large size and KITTI is insufficiently large to train it. For models that are well trained after running only on data augmented KITTI, do they scale poorly to larger datasets because they are able to saturate on just KITTI? How well do non-Uber ATG developed methods such as VoxelNet, SECOND, or PointPillars perform on ATG4D relative to LaserNet?

4. The paper provides very little data on how much each network component contributes to the total inference time, indicating only how much time feed forward inference took and how much time total processing took. What components of the network can be changed in order to most directly reduce inference time?

4 Gaps in the Literature when Building an Intelligent Agent and Directions of Future Research

Imagine you have just been made the lead architect for the vision and navigation system for a campus-wide indoor/outdoor courier robot. Your robot needs to be able to navigate to waypoints on the map, cross the street without being hit by cars, and avoid cutting in front of pedestrians. In order to execute on these requirements safely and effectively, your system needs to be able to detect and predict the trajectories of other agents such as cars and pedestrians and be highly responsive to changes in these trajectories in order to avoid collisions. Worse, due to spotty WiFi and low latency requirements, all core computation needs to be done on-board, forcing you to contend with limited computational resources (limited CPU, GPU, and memory). With these constraints in mind, you turn to the literature outlined above; chances are, you're greatly dismayed. However, these shortcomings highlight new or burgeoning directions of fruitful research.

Each method presented has a different runtime and prediction quality; comparing KITTI mean AP results does not represent the full set of constraints faced by your robot. For example, in the literature each bounding box match is evaluated against the point cloud given to the method as input. However, in the real world, the environment is changing while the method is running inference. If the method spends a long time computing high-quality bounding boxes, by the time the bounding boxes are output, the world may have changed so much that they are rendered useless. This problem has recently been raised in the image

segmentation domain and a new accuracy metric, streaming AP [22], has been proposed to address it. Streaming AP operates over timestamped sequences of observations, providing the first observation to the method and, when it finishes inference, evaluating the output against the most recent prior observation in the sequence relative to its inference completing time rather than the input observation. This evaluation allows for practitioners to construct and explore the Pareto frontier between runtime and quality via a single quantitative score.

In a similar vein, methods currently execute and then produce a single result. If the robot is going slowly or if it needs to perform a delicate maneuver near an object, it may want a higher quality solution. Developing *anytime* approaches to object detection would allow for the system to not only make a binary tradeoff, but employ online metareasoning [23] to fine-tune this runtime/quality tradeoff. IBB-Net [24] serves as early work in the space of anytime detectors, using RPNs to broadly identify regions of objects and then using iterative applications of a PointNet-based bounding box regressor to iteratively refine bounding box predictions.

Most methods also ignore the realities of the sensors used to generate the point cloud. As LIDAR can operate outdoors in direct sunlight and at night with no change in performance, it is the most popular point cloud sensor; however, most wide view LIDAR use a vertically stacked array of lasers that continuously rotate, taking readings of vertical slices at fractions of every degree to produce a new global scan at approximately 10Hz. As a direct result, a global scan is not taken with a global shutter; data is streamed off the sensor as it is collected. Thus, on a 10Hz sensor, data taken at the beginning of a global scan is approximately 100 milliseconds older than data taken at the end. As far as we are aware, there is not any literature addressing this issue, but there are several straight-forward strategies. For instance, rather than waiting to process the entire point cloud, the method could process small wedges as they are streamed off the sensor. This approach has the added benefit of requiring a smaller amount VRAM to store each wedge and a smaller scene area to consider for grid-based approaches, thus providing an opportunity to further shrink network size and increase inference speed; however, experiments would need to be done to determine the extent of the potential detection quality degradation.

In a similar vein, methods currently pay equal attention to all parts of the scene when looking to detect objects; however, the robot has additional semantic information about important areas in the scene from sources such as the map, allowing it to know to pay attention to special areas such as roads, sidewalks, or blind corners. Rather than pay equal attention to all parts of the scene, the detection system could reduce attention to unimportant areas or consider important areas at higher resolution. For example, VoteNet [25] explicitly performs sampling for location diversity; this sampling could be biased by attention in order to only process important areas. Alternatively, an approach similar to Differentiable Patch Selection [26], an image based object detector, could allow for the scene to be broadly processed at a low resolution, with important regions mined using it differentiable-k selection approach then process at full resolution.

Finally, each of these methods also consider themselves in isolation. On the real robot, other systems such as the localization system and the tracking system are also operating over observations from the robot's sensors. If methods were to employ a single shared backbone used by multiple systems, it would provide better synchronization between systems (it's less likely one system will have some knowledge about the scene unavailable to another system) and allow for a single backbone that shares compute and memory budget allocated to all systems, allowing the backbone to be bigger. Additionally, the stated object detection problem formulation is to detect an object in a single snapshot of the environment; in many

cases, observations from immediately prior and information from the robot's object tracking framework are readily available and provide additional sources of data to reason about the scene, but they are ignored. Coupling this information into the system can enable better performance and enable direct forecasting of object movement as part of prediction, akin to recent work in panoptic image segmentation [27].

5 Bibliography

References

[1] A. Geiger, P. Lenz, R. Urtasun, Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite, in: Conference on Computer Vision and Pattern Recognition (CVPR), 2012.

[2] I. Armeni, O. Sener, A. R. Zamir, H. Jiang, I. Brilakis, M. Fischer, S. Savarese, 3D Semantic Parsing of Large-Scale Indoor Spaces, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 1534–1543.

[3] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, F. Yu, ShapeNet: An Information-Rich 3D Model Repository, Tech. Rep. arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago (2015).

[4] M. Everingham, L. Gool, C. K. Williams, J. Winn, A. Zisserman, The Pascal Visual Object Classes (VOC) Challenge, Int. J. Comput. Vision 88 (2) (2010) 303–338.

[5] M.-F. Chang, J. W. Lambert, P. Sangkloy, J. Singh, S. Bak, A. Hartnett, D. Wang, P. Carr, S. Lucey, D. Ramanan, J. Hays, Argoverse: 3D Tracking and Forecasting with Rich Maps, in: Conference on Computer Vision and Pattern Recognition (CVPR), 2019.

[6] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. 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, 2020, pp. 2446–2454.

[7] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, O. Beijbom, nuScenes: A multimodal dataset for autonomous driving, arXiv preprint arXiv:1903.11027 (2019).

[8] R. Charles, H. Su, K. Mo, L. Guibas, PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation, 2017, pp. 77–85.

[9] Y. Zhou, O. Tuzel, VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection, 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (2018) 4490–4499.

[10] Y. Yan, Y. Mao, B. Li, Second: Sparsely embedded convolutional detection, Sensors (Basel, Switzerland) 18 (2018).

[11] B. Graham, M. Engelcke, L. van der Maaten, 3D Semantic Segmentation with Submanifold Sparse Convolutional Networks, CVPR (2018).

[12] B. Graham, L. van der Maaten, Submanifold Sparse Convolutional Networks, arXiv preprint arXiv:1706.01307 (2017).

[13] S. Ren, K. He, R. Girshick, J. Sun, Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks, IEEE Transactions on Pattern Analysis and Machine Intelligence 39 (6) (2017) 1137–1149.

[14] A. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, O. Beijbom, Pointpillars: Fast encoders for object detection from point clouds, 2019, pp. 12689–12697. doi:10.1109/CVPR.2019.01298.

[15] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, A. Berg, SSD: Single Shot MultiBox Detector, Vol. 9905, 2016, pp. 21–37.

[16] G. P. Meyer, A. Laddha, E. Kee, C. Vallespi-Gonzalez, C. Wellington, LaserNet: An Efficient Probabilistic 3D Object Detector for Autonomous Driving, 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2019) 12669–12678.

[17] K. He, X. Zhang, S. Ren, J. Sun, Deep Residual Learning for Image Recognition, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770–778.

[18] E. Rublee, V. Rabaud, K. Konolige, G. Bradski, ORB: An efficient alternative to SIFT or SURF, in: 2011 International Conference on Computer Vision, 2011, pp. 2564–2571.

[19] R. Mur-Artal, J. D. Tardós, ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras, IEEE Transactions on Robotics 33 (5) (2017) 1255–1262.

[20] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, S. Belongie, Feature Pyramid Networks for Object Detection (12 2016).

[21] Yizong Cheng, Mean Shift, Mode Seeking, and Clustering, IEEE Transactions on Pattern Analysis and Machine Intelligence 17 (8) (1995) 790–799.

[22] M. Li, Y. Wang, D. Ramanan, Towards Streaming Perception, in: ECCV, 2020.

[23] J. Svegliato, P. Sharma, S. Zilberstein, A Model-Free Approach to Meta-Level Control of Anytime Algorithms, in: Proceedings of the International Conference on Robotics and Automation, 2020.

[24] B. Miller, IBB-Net: Fast Iterative Bounding Box Regression for Detection on Point Clouds, Master's thesis, Pittsburgh, PA (June 2020).

[25] C. R. Qi, O. Litany, K. He, L. Guibas, Deep Hough Voting for 3D Object Detection in Point Clouds, in: 2019 IEEE/CVF International Conference on Computer Vision (ICCV), 2019, pp. 9276–9285.

[26] J.-B. Cordonnier, A. Mahendran, A. Dosovitskiy, D. Weissenborn, J. Uszkoreit, T. Unterthiner, Differentiable Patch Selection for Image Recognition, CVPR (2021).

[27] C. Graber, G. Tsai, M. Firman, G. Brostow, A. Schwing, Panoptic Segmentation Forecasting (2021). arXiv:2104.03962.

A PointNet

PointNet, as formulated in its original paper [8], is a point based approach that takes in an unordered point cloud and produces both a single k-class classification for the entire point cloud and semantic label for each point in the point cloud; examples are shown in Figure 10.

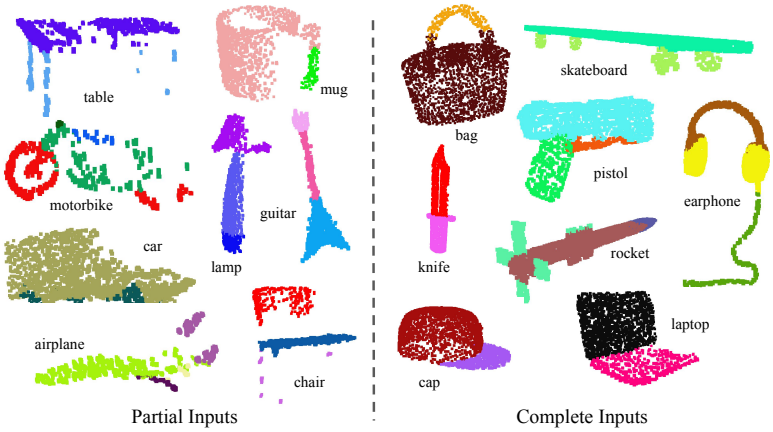


Figure 10: Example results from the PointNet [8] paper.

At a high level, PointNet operates via a backbone that takes in the raw points and performs identical operations on each, with the end of the pipeline collapsing each point's embedding into a single global feature vector via max-pooling. This global feature vector is then fed into a classification head to predict the point cloud's class, and this global feature vector is concatenated to each point in an earlier per-point embedding and then fed into a segmentation head to perform per-point segmentation. This architecture is shown in Figure 11.

The backbone of PointNet is primarily composed of Multi-Layer Perceptrons (MLP) and two Transformation Networks (T-Nets), each applied in parallel to each of the n input points. The MLPs are used to project the features into higher dimensional spaces, and the T-Nets are used normalize the pose of the input object, as the underlying object can be presented in a variety of poses. The first T-Net, run on the raw input, regresses a 3×3 matrix which performs a linear transformation on all points in order to put the object into a canonical form. The second T-Net performs a similar regression on the latent space representation producing a 64×64 transform matrix. As the second T-Net is a much larger and thus a more difficult optimization problem, its output is regularized to be orthogonal. Of note, the final layer of the backbone is a max-pool across the $n \times 1024$ latent space to form a 1×1024 global feature vector. Max-pool is symmetric, i.e. order of inputs does not matter, allowing the backbone to maintain its order invarance.

To validate their design decisions, the authors performed three ablative tests: 1) alternative means of handling order, 2) alternative symmetric functions, and 3) ablation of the T-Nets from their architecture. For alternative order handling, the authors consider an

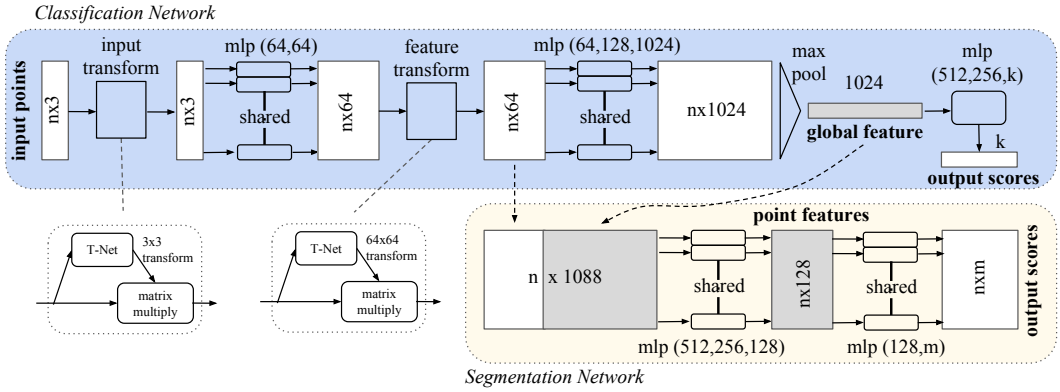


Figure 11: Architecture diagram from PointNet [8] paper. Many approaches including VoxelNet [9] and PointPillars [14] use modified versions of PointNet's backbone to produce its global feature as a point vectorizer.

LSTM which treats the input points as a sequence, as well as an MLP that takes unordered or canonically ordered points; PointNet outperforms both the MLP based approach (factor of 2) and the LSTM (approximately 9%). For alternative symmetric functions besides max pool, the authors consider sum and average, but max-pool produced 4% better accuracy relative to both. For ablation of the T-Nets, the authors consider removal of both T-Nets, just the 3×3 T-Net, just the 64×64 T-Net without regularization, and just the 3×3 T-Net with regularization; the results show approximately 2% performance increase in classification accuracy using both T-Nets over none. With the T-Nets, the model has 3.5M params and takes 440M FLOPs/sample, while without the T-Nets, the model is 0.8M params and takes 148M FLOPs/sample.

PointNet is an attractive vectorizer because, without the T-Nets, it is small, order agnostic, and able to encode a wide variety of holistic information in its global feature. Importantly, many approaches claim to use a PointNet vectorizer (e.g. PointPillars), but propose significant changes to the dimensionality, layer count, and non-linearities used in its backbone; the common connection between these networks is the general behavior of performing shared transformations over each point and then performing max pooling to generate a global feature vector.