LidarNAS: Unifying and Searching Neural Architectures for 3D Point Clouds

Chenxi Liu, Zhaoqi Leng, Pei Sun, Shuyang Cheng, Charles R. Qi, Yin Zhou, Mingxing Tan, and Dragomir Anguelov

Waymo LLC
{cxliu, lengzhaoqi, peis, shuyangcheng, rqi, yinzhou, tanmingxing, dragomir}@waymo.com

Abstract. Developing neural models that accurately understand objects in 3D point clouds is essential for the success of robotics and autonomous driving. However, arguably due to the higher-dimensional nature of the data (as compared to images), existing neural architectures exhibit a large variety in their designs, including but not limited to the views considered, the format of the neural features, and the neural operations used. Lack of a unified framework and interpretation makes it hard to put these designs in perspective, as well as systematically explore new ones. In this paper, we begin by proposing a unified framework of such, with the key idea being factorizing the neural networks into a series of view transforms and neural layers. We demonstrate that this modular framework can reproduce a variety of existing works while allowing a fair comparison of backbone designs. Then, we show how this framework can easily materialize into a concrete neural architecture search (NAS) space, allowing a principled NAS-for-3D exploration. In performing evolutionary NAS on the 3D object detection task on the Waymo Open Dataset, not only do we outperform the state-of-the-art models, but also report the interesting finding that NAS tends to discover the same macro-level architecture concept for both the vehicle and pedestrian classes.

1 Introduction

Being able to recognize, segment, or detect objects in 3D is one of the fundamental goals of computer vision. In this paper we consider the point cloud input representation for the wide usage of RGBD cameras in robotics applications, as well as LiDAR sensors in autonomous driving. There has been a lot of research in this area, including various deep learning based approaches.

But which neural architecture should you choose? PointNet [32]? VoxelNet [55]? PointPillars [18]? Range Sparse Net [43]? It is easy to get overwhelmed by the diverse set of concepts present in these names as well as the variety in the architectures themselves.

This level of variety at the macro-level is not observed in other areas, e.g., neural architectures developed for 2D images. The root cause is the higher-dimensional nature of the data. There are three major reasons in particular:

- *Views*: 2D images are captured by an egocentric photographer. A similar view exists for 3D, that is the perspective view, or range images. But when the scan is not egocentric, we have an unordered point set that can no longer be indexed by pixel coordinates. In addition, gravity makes the z axis special, and often times a natural choice is to view an object from top-down. Each view has its unique properties and (dis)advantages.
- *Sparsity*: Images are dense in the sense that each pixel has an RGB value between 0 and 255. But in 3D, range images may have pixels that correspond to infinite depth. Also, objects typically occupy a small percentage of the space, meaning that when a scene is voxelized, the number of non-empty voxels is typically small compared with the total number of voxels.
- *Neural operations*: Due to views and sparsity, 2D convolution does not always apply, resulting in more diverse neural operations.

Our first contribution in this paper is a *unified framework* that can interpret and organize the variety of neural architecture designs, while adhering to the principles listed above. This framework allows us to put existing designs in perspective and enables us to explore new designs. The key idea is to factorize the entire neural network into a series of *transforms* and *layers*. The framework supports four views (point, voxel, pillar, perspective) and two formats (dense, sparse), as well as the *transforms* between them. It is also possible to merge features from different views, building parallelism into the sequential stages. But once a view-format combination is set, it restricts the types of *layers* that can be applied. When visualized, this framework is a trellis, and any neural architecture corresponds to a connected subset of this trellis. We provide several examples of how popular architectures can be refactored and reproduced under this framework, proving its generality.

A direct benefit of this framework is that it can easily materialize into a search space, which immediately unlocks and enables NAS. NAS stands for neural architecture search [57], which tries to replace human labor and manual designs with machine computation and automatic discoveries. Despite its success on 2D architectures [44], its usage on 3D has been limited. In this paper we conduct a principled NAS-for-3D explorations, by not only considering the micro-level (such as the number of channels), but also embracing the macro-level (such as transforms between various views and formats).

We conduct our LidarNAS experiments on the 3D object detection task on the Waymo Open Dataset [42]. Using regularized evolution [35], our search finds LidarNASNet, which outperforms the state-of-the-art RSN model [43] on both the vehicle and the pedestrian classes. In addition to the superior accuracy and the competitive latency, there are also interesting observations about the LidarNASNet architecture itself. First of all, though the search / evolution was conducted separately on vehicle and pedestrian, the found architectures have essentially the same high-level design concept. Second, the modifications discovered by NAS coincidentally reflects ideas from human designs. We also analyze the hundreds of architectures sampled in the process and draw useful lessons that should inform future designs.

To summarize, the main contributions of this paper are:

- A unified framework general enough to include a wide range of backbones for 3D data processing
- A search space and an algorithm challenging enough to cover both the micro-level and the macro-level
- A successful NAS experiment which leads to state-of-the-art performance on the Waymo Open Dataset

2 Related Work

2.1 Neural Architectures for 3D

We partition neural architectures for 3D into four categories, according to the primary view(s) used. Since this paper studies *backbone* design for 3D object detection, we will mostly cover detection but will also talk about segmentation and classification.

The first category is **top-down primary**, which includes voxel and pillar. The main idea is to divide 3D points into 3D voxels [11, 8, 50, 55, 49, 9] or 2D pillars [18], which then become regular. The advantage is that voxelization enables locality, which in turn enables convolution operations. But the main limitation is memory consumption, which grows cubically (or quadratically). This either limits the maximum detection range or sacrifices the voxelization granularity. Even if sparse operations may be used, for egocentric scans, the point densities at long-range and short-range are different, posing challenges in learning.

The second category is **point primary**, which treats the point cloud as unorganized sets. Originally developed for classification and segmentation [32, 33], the idea can also be used on detection [31, 29]. The advantage is that it is more memory-friendly than voxelization based approaches. However, its limitation is that the neural layers do not perform as well, possibly due to irregular coordinates. In addition, to achieve locality, nearest neighbor search is typically needed for the input, which can be expensive.

The third category is **perspective primary**, operating directly on the range image [28, 5, 6, 12]. This is also very memory-friendly and can utilize powerful 2D convolution layers which have been extensively researched. However, as the depth can change drastically for adjacent pixels, these methods exhibit more difficulty in localizing the objects accurately, as well as handling occlusions.

The fourth and final category is **fusion** methods, which use two or more of the representations discussed above. The fusion may be either sequential and parallel. For example, RSN [43] sequentially performs foreground segmentation on the perspective view and delivers detection output on the top-down view. PVCNN [25] and SPVCNN [45] fuses information from the point view and the voxel view in a parallel fashion. MVF [54] fuses feature from perspective view, point view, and pillar view, also in a parallel fashion. The hope is that fusion methods can combine the best of multiple worlds, which is why it is important to keep all options when doing architecture exploration.

2.2 Neural Architecture Search

Early works on neural architecture search primarily focused on the **search algorithm**. A variety of methods were introduced, including reinforcement learning [57, 3], evolution [36, 35], performance prediction [23], weight-sharing [30, 24]. Essentially, different methods make different approximations about the search process.

These search algorithm explorations started on image classification. The following phase consists of extending to other **tasks**, such as semantic segmentation [7, 22] and object detection [48, 14]. For 3D tasks, NAS research has been done on medical imaging [56, 17, 2, 47, 52]. However, the *volumetric* CT scans are different from *point* clouds, and as a result the search space is greatly simplified. There are also works on 3D shape classification [26, 19], but their overall frameworks do not exceed that set by [24]. [45, 20] is closer to our work, in the sense that it uses NAS to optimize for segmentation and detection on 3D scenes (KITTI [13]). But generalizing the terminology used in [22], we believe there is also a two-level hierarchy in 3D neural architecture designs, with the outer macro-level controlling the views of the data / features, and the inner micro-level being the specifics of the neural layers. Under this terminology, [45, 20] keeps the macro-level fixed, while our search covers both.

3 Unifying Neural Architectures for 3D

3.1 Philosophy

In order to offer a unified interpretation of the growing variety of neural networks for 3D, we need to pinpoint their high-level design principles. Fortunately, we find these underlying principles to be surprisingly congruent, and we characterize them as: finding *some neighborhood* of the 3D points and then *aggregating information* within. The "aggregation" part is typically done through some form of convolution and / or pooling. The "neighborhood" part has different choices:

- PointNet [32]: the neighborhood alternates between the point itself (MLP) and all points (max-pooling)
- PointNet++ [33]: the neighborhood is an Euclidean ball with a certain radius
- VoxelNet [55]: 3D neighborhood measured by Manhattan distance of Cartesian coordinates (x, y, z)
- PointPillars [18]: 2D neighborhood measured by Manhattan distance of (part of) Cartesian coordinates (x, y)
- LaserNet [28]: 2D neighborhood measured by Manhattan distance of pixel coordinates (i, j)

These common "neighborhood" choices have been typically expressed through the views of the data / features: point, voxel, pillar, perspective. We point out that there have been and will be more views being proposed, which is why we feel the "neighborhood" interpretation is more generic. Notably, different data views can *transform* between each other back and forth. However, once the data

view is determined, it *restricts* the type of *layers* that can be applied. This factorization of "transforms" and "layers" as well as their relationship will be reflected in our framework described next.

3.2 A Unified Framework

In this subsection, we build upon the aforementioned high-level ideas and describe the main framework we use to think about neural architectures throughout this work. We describe its different levels of detail from fine to coarse.

Views and formats We consider a total of four views (point, pillar, voxel, perspective) and up to two data formats (dense and sparse):

- **Point**: The features for all N 3D points are stored in a matrix of size $[N, C]$, where C is the number of channels. The Cartesian coordinates (x, y, z) for each point are stored in a separate matrix of size $[N, 3]$, where the indices of the points are aligned between the two matrices.
- **Pillar**: In this view, we store a fixed-length feature for each pillar when viewing the scene from top-down. We allow the pillar view to be either dense or sparse. If dense, the features are stored in a tensor of size $[B, X, Y, C]$, where B is the batch size, X and Y are the number of pillars along the corresponding dimension. If sparse, the features are stored in a matrix of size $[N, C]$, where N is the number of *non-empty* pillars and a separate matrix of size $[N, 3]$ is used to store the indices (both batch and spatial) of these non-empty pillars. In both data formats, unlike the point view, the Cartesian coordinates of each pillar('s center) can be easily calculated from its spatial index (`origin + index * pillar size`).
- **Voxel**: Different from the pillar view, the voxel view partitions the scene along all three spatial dimensions. The additional partition along the z axis makes fitting a tensor of size $[B, X, Y, Z, C]$ into memory very challenging. Therefore, in this work we only consider the sparse format for the voxel view. Features are stored in a matrix of size $[N, C]$, where N is the number of *non-empty* voxels. A separate matrix of size $[N, 4]$ is used to store their indices (both batch and spatial).
- **Perspective**: For egocentric 3D scans or RGBD images, simply using the original perspective view is a natural choice. We consider both the dense and sparse formats for this view. If dense, features are stored in a tensor of size $[B, H, W, C]$, where H and W consist of the size of the range image. A separate tensor of size $[B, H, W, 3]$ is used to store the Cartesian coordinates of each pixel on the range image. If sparse, features are stored in a matrix of size $[N, C]$ and Cartesian coordinates a separate matrix of size $[N, 3]$, similar to the point view.

Transforms Now that the views and formats are established, the framework shall be general enough to include possible transforms from one to the other. The transforms are intended to be lightweight: powerful neural feature update is

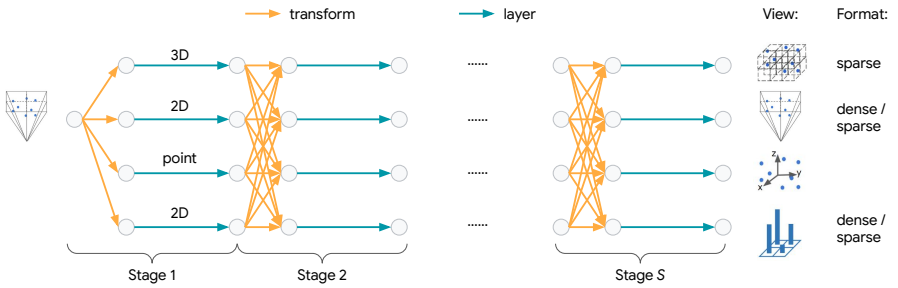


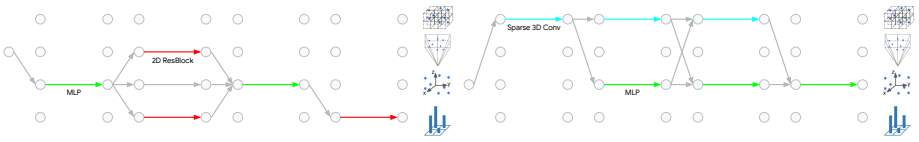
Fig. 1: The LidarNAS framework for interpreting neural architectures on 3D point clouds. The entire backbone consists of S stages. Each stage consists of view & format transforms followed by corresponding neural layers. Within this framework, a backbone architecture corresponds to a *connected subset* of the S-stage trellis.

a non-goal. Since there are totally six possible representations (four views, with pillar and perspective having two formats), we have up to $6^2 = 36$ different transforms. Though this number may seem daunting, some of these transforms have more familiar and friendly names. For example, the transform to itself is identity. The one from a sparse format to its dense counterpart is densification (by padding zero vectors). The point to voxel transform is voxelization. The reverse transform is devoxelization. The point to perspective transform is projection.

Layers Once a view-format combination is set, we can apply neural layers to update the features. Generally, we do not put constraint on the number or form of the layers: it can be as simple as a one-layer convolution, or as complicated as an entire U-Net [37]. But the one constraint is that it conforms to the view-format combination for both its input and output. This is because, for instance, 2D convolution cannot be applied on 3D inputs; sparse convolution does not work on dense features. Notably, 2D layer implementations can interchangeably work for both the pillar view and the perspective view.

Stages Putting these concepts together, we define a stage to be the sequential pair of possible transforms and their associated layers. Fig. 1 visualizes the concatenation of S stages. Within this framework, the backbone of a neural network for 3D corresponds to a *connected subset of this S-stage trellis*. A head can then be added to the end to perform 3D classification / detection / segmentation.

We emphasize that the word choice is "subset" but not "path", meaning that a stage can have more than one view present. This makes our framework more general, as it supports not only sequential designs but also parallel ones. Consequently, we may have multiple different views in stage $s-1$ transforming to the same view in stage s. In these cases, after applying individual transforms, we merge these transformed features through either concatenation (default in this work) or summation.



(a) Multi-View Fusion [54] (b) Sparse Point-Voxel [45]

Fig. 2: Examples of how existing designs may be interpreted within the LidarNAS framework.

3.3 Inclusion of Existing Designs

In Fig. 2 and Fig. 3, we visualize several examples of how existing designs may be interpreted within the framework described above. Our framework is flexible enough to cover both entirely sequential designs such as Range Sparse Net [43] and more parallel designs such as Multi-View Fusion [54]. In addition to these networks developed for 3D detection, it can also explain those beyond, such as SPV [45]. More architecture designs fit in, including but not limited to [55, 49, 18, 29, 46], but we skip visualization due to space limitations.

4 Searching Neural Architectures for 3D

The framework described in the previous section brings many benefits, one of which is the potential to search novel and better architectures. This section focuses on how the framework materializes into a search space (Sec. 4.1), as well as our choice of search algorithm (Sec. 4.2).

4.1 From Framework to Search Space

Fig. 2 demonstrated how, at a high level, various architectures fall within the LidarNAS framework. But delving into the details, specific implementations of the modules are going to differ across works. While this is very much expected and understandable, the variety and freedom in "layers" alone would make constructing a meaningful search space infeasible. We now discuss how we materialize the framework into a search space by making specific choices.

Transforms Among the 36 possible transforms, we did not implement the transforms from pillar to voxel, as nothing more can be done other than copying the same features along the z axis. From the voxel view, our implementation only supported transforms to the pillar view. Supporting 31/36 transforms is still high coverage.

Layers We need at least one type of neural layer for each of the following: point, 2D dense, 2D sparse, 3D sparse. Our search space picked one representative for each:

- **Point**: Multiple layers of dense-normalization-ReLU. The normalization can either be batch normalization [16] or layer normalization [1]. The number of units F in the dense layers is a hyperparameter that can be searched.
- **2D dense**: A U-Net [37] with residual blocks [15]. We use up to five downsampling and upsampling scales. The number of channels for each scale are $[F, 4F, 8F, 8F, 16F]$ with F being the hyperparameter that can be searched. The number of blocks per scale is 2 except for the highest resolution scale which is 1.
- **2D sparse**: Also a U-Net with residual blocks, except that each convolution is a sparse convolution (kernel size 3×3). We use up to 3 downsampling and upsampling scales, and the number of blocks are $[1, 2, 3]$ and $[0, 2, 2]$. We use the same number of channels F for all downsampling and upsampling blocks.
- **3D sparse**: Also a U-Net with residual blocks, except that 3D sparse convolution is used. The kernel size can either be $3 \times 3 \times 3$ or $3 \times 3 \times 1$, and the corresponding stride for each scale is $2 \times 2 \times 2$ or $2 \times 2 \times 1$. The other details follow the 2D sparse case above.

These choices of layer specifics, especially those for 2D and 3D, try to exactly follow RSN [43].

Stages Our search space considers $S = 3$ stages. For simplicity, we also have the constraint that the last stage can only have one view. Inspired by RSN, we add the option to perform foreground segmentation immediately after the first perspective branch that appears.

4.2 Regularized Evolution

We choose regularized evolution to be our search algorithm, which follows [35]. Compared against other major classes of NAS methods, evolution arguably makes the least amount of approximations, which is desirable especially since we are exploring a less explored task and a complicated search space. We do not use weight-sharing NAS for GPU memory considerations. 3D tasks are understandably more memory intensive than 2D tasks, and the batch size on each GPU was already small (< 10). However, even the best weight-sharing NAS (a recent example is [4]) implementations require $2 - 3\times$ extra GPU memory.

Our mutation algorithm works by first randomly selecting a stage s and then randomly applying one of the following six mutation choices to this stage:

- *Add a view*: if the stage does not have all four views, then randomly add a view not yet present in this stage. A random view present in the previous stage is selected as its predecessor. A random view present in the next stage is selected as its successor. A default layer of the corresponding type is used for this addition. The number of channels for all layers in this stage are halved.

- *Remove a view*: if the stage has more than one view, then randomly remove an existing view. Usage of the removed view in the next stage is also removed. The number of channels for all layers in this stage are doubled.
- *Switch the view*: if the stage has exactly one view, then switch the view to another. All usage of the old view in the next stage is changed to the new view.
- *Adjust the pillar / voxel size*: a key parameter in many of the transforms is the pillar / voxel size. Multiply the pillar / voxel size by either 0.8 or 1.2 for all views.
- *Adjust the number of channels*: multiply the number of channels for all layers in the stage by either 0.8 or 1.2.
- *Adjust the layer progression*:
  - Point: Either increase or decrease the number of dense-normalization-ReLU by 1.
  - 2D dense: Either increase or decrease the number of scales by 1.
  - 2D / 3D sparse: Increase or decrease the number of downsampling / upsampling scales by 1.

If a mutation fails (e.g., if the precondition does not hold, such as trying to remove a view when the stage only has one view), the algorithm mutates again until it succeeds.

The first four mutation choices focus on the "transform" aspect of a stage, while the last two mutation choices focus on the "layer" aspect. This level of coverage and variety makes the search comprehensive yet challenging.

5 Experimental Results

5.1 Experimental Setting

We perform 3D object detection experiments on the challenging Waymo Open Dataset [42]. It provides LiDAR scans in the range image form, which makes experiments on the perspective view much more natural and convenient. It contains 1150 LiDAR sequences with 798 train, 202 validation, and 150 test ones. Each sequence is 20 seconds at 10 frames per second. Experiments are conducted on both the vehicle and the pedestrian classes, using the official evaluation metrics of 3D / BEV AP.

5.2 Existing Architectures under LidarNAS

In this subsection, we use the LidarNAS framework (Sec. 3) to reimplement several existing neural architectures for 3D. The goal here is to prove the generality and correctness of the LidarNAS framework interpretation, as well as validate our implementation of individual modules.

We selected four existing architectures: RSN [43], PointPillars [18], LaserNet [28], and MVF++ [34]. These are selected to cover a variety of views as well as topology. Note that the LidarNAS framework focuses on *backbone* design. In

model	class	frame	device	batch	steps	lr	voxelization (m)	LidarNAS AP	previous AP
RSN-exact	Veh	3	GPU	2×16	120k	0.006	$0.2 \times 0.2 \times 0.2$	77.2	77.2 [43]
	Ped	3	GPU	3×16	120k	0.006	$0.1 \times 0.1 \times 20.0$	79.1	79.1 [43]
PointPillars-like	Veh	1	GPU	2×16	120k	0.006	0.32×0.32	69.3	63.3 [43] / 60.3 [34]
	Ped	1	GPU	3×16	120k	0.006	0.32×0.32	66.1	68.9 [43] / 60.1 [34]
LaserNet-like	Veh	1	GPU	1×16	360k	0.001	-	47.1	52.1 [43] / 56.1 [6]
	Ped	1	GPU	1×16	240k	0.003	-	59.0	63.4 [43] / 62.9 [6]
MVF++-like	Veh	1	TPU	2×128	43k	0.003	0.32×0.32	73.6	74.6 [34]
	Ped	1	TPU	2×128	43k	0.003	0.32×0.32	70.4	78.0 [34]

Table 1: A diverse set of existing 3D detection architectures under the LidarNAS framework. The second number in the batch size multiplication is the number of GPUs / TPU shards. The metric (last two columns) is L1 3D AP.

our reimplementation, we use an anchor-free detection *head* that is the same as RSN if the backbone output is sparse voxels but also works for pillar and perspective views (details are described in the supplementary material). This means that our RSN reimplementation is exact but the others are not, and we add the suffix "-like" to indicate this difference.

Tab. 1 summarizes the results, using L1 3D AP. Key hyperparameter values are also provided. We use no color if our reimplementation is within 1% absolute of the previously reported number; green if higher than $> 5\%$; yellow if lower than $\leq 5\%$; and red if lower than $> 5\%$. Considering the diversity of these architectures, overall we consider our reimplementation to be acceptable and successful, validating our implementation of (some of the) transforms and layers modules. Notice that our implementation can support multi-frame, as well as both GPU and TPU.

Looking into individual neural architectures, our reproduction of RSN is exact. Interestingly, PointPillars-like significantly outperforms previous reports on the vehicle class. This is an important reminder that revisiting previous architectures may be necessary and beneficial, as they may still be competitive when coupled with latest developments in other areas (e.g., anchor-free detection head). However, the performance on the pedestrian class is slightly worse. This is also observed on MVF++-like, where the vehicle class is within 1% but the pedestrian class is significantly worse. Our hypothesis is that comparatively speaking, our detection head is better suited on larger objects but struggles more on smaller objects. Finally, our LaserNet-like performs noticeably worse than any network that detects on the top-down view (meaning pillar or voxel), despite training for $2 - 3\times$ longer steps. This proves that detection from the perspective view needs more specialized operations, such as those described in the original paper, or some recent developments [6, 12].

5.3 Searching for New Architectures

In this subsection, we perform and analyze neural architecture search experiments, using the search space and algorithm described in Sec. 4.

model	year	frame	Vehicle 3D AP	Vehicle BEV AP	Vehicle latency	Pedestrian 3D AP	Pedestrian BEV AP	Pedestrian latency
LaserNet [28]	CVPR 19		52.1	71.2	64.3	63.4	70.0	64.3
PointPillars [18]	CVPR 19		63.3	82.5	49.0	68.9	76.0	49.0
PV-RCNN [38]	CVPR 20		70.3	83.0	-	-	-	-
Pillar-based [46]	ECCV 20	1	69.8	87.1	66.7	72.5	78.5	66.7
PV-RCNN [39]	WOD 20	2	77.5	-	300	78.9	-	300
RCD [5]	CoRL 20	1	69.0	82.1	-	-	-	-
MVF++ [34]	CVPR 21	1	74.6	87.6	-	78.0	83.3	-
CenterPoint [51]	CVPR 21	2	76.7	-	-	79.0	-	-
PPC [6]	CVPR 21		65.2	80.8	-	75.5	82.2	-
RangeDet [12]	ICCV 21	1	72.9	-	-	75.9	-	-
PointPillars-like[§]		1	67.6	85.3	-	-	-	-
LidarNASNet-P (ours)		1	**73.2**	**88.2**	-	-	-	-
RSN [43]	CVPR 21	1	75.2	87.7	46.5[†]	77.1	81.7	21.0[†]
LidarNASNet-R (ours)		1	**75.6**	**88.6**	49.3[†]	**77.4**	**82.0**	22.6[†]

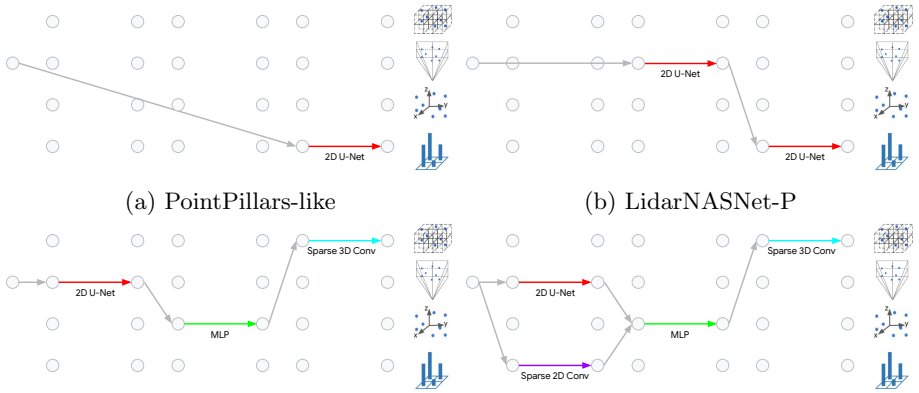
Table 2: 3D object detection results for the vehicle and pedestrian classes on the Waymo Open Dataset validation set. The AP is difficulty L1. The unit of latency is ms. Multi-frame models are grayed. [§]: Slightly different from PointPillars-like in Tab. 1, because here we have to swap the original layers with the U-Net explained in Sec. 4.1. [†]: our measurement using identical setting: average on 10 scenes, each has more than 100 vehicles / pedestrians.

Evolving past the state-of-the-art Based on the analysis above, picking a random architecture as the starting point would take much longer time for the performance to ramp up, so we use *warm starting* [40, 41] to speed up and save up. Each search lasts 100 architectures, each trained using batch size 2×8 GPUs for 12k steps (10% of the standard number of steps) using cosine learning rate. All architectures operate on single-frame. The population size and tournament size for the regularized evolution algorithm are 20 and 5 respectively. We also measure the V100 latency of the network on a (random) training batch immediately after 11k training steps. The measurement is taken close to the end of the training because for architectures that perform foreground segmentation, the latency may change throughout training. We comment that this search phase latency measurement is noisy, not only because the data batch is random, but also because the scheduler may allocate a GPU shared with other jobs. Regardless, we use `100 * L1 3D AP - 0.5 * latency in ms`[1] as the objective to guide the evolution. Once an architecture is identified, we increase the per-GPU batch size from 2 to 5, and train for 120k steps as the final evaluation.

We conduct a separate search / evolution from three different starting points: PointPillars-like vehicle, RSN CarXL, and RSN PedL[2]. We name our found architecture LidarNASNet-P / R depending on whether the starting point was PointPillars-like or RSN, and compare them against other models in Tab. 2.

[1] We empirically picked these multipliers; did not tune them heavily.

[2] We skipped PointPillars-like pedestrian, because the corresponding number in Tab. 1 is yellow not green.



(a) PointPillars-like (b) LidarNASNet-P

(c) Range Sparse Net [43] (d) LidarNASNet-R

Fig. 3: The *macro-level* architecture of the found LidarNASNet-P / R. Note that the illustration in (d) applies for *both* vehicle and pedestrian. It adds a sparse convolution on the pillar view in the first stage, utilizing all four views and two formats considered in this work.

We first compare LidarNASNet-P against PointPillars-like, the evolution baseline. The L1 3D AP improves from 67.6 to 73.2, with a significant gap of +5.6. The gain on BEV AP is also significant at +2.9. The large improvement and competitive end result clearly showcase the effectiveness of our search. When running the same evolution from RSN, LidarNASNet-R outperforms by 0.4 and 0.3 3D AP on vehicle and pedestrian respectively, and the gains on BEV AP are even larger. As we will see soon, LidarNASNet-R has an additional branch, so the latency is higher, but only slightly. To put this in perspective, we did an ablation study[3] where we increase the number of channels in the sparse U-Net of RSN for vehicle (from 64 to 91) to reach AP parity with LidarNASNet-R. The latency of this architecture is 60.8ms, which is significantly higher.

Comparing against other architectures, LidarNASNet-R also performs very competitively. Not only is this reflected in the superior AP especially among single-frame models, but also in the small latency. We reiterate that our total search cost is about 80 GPU days, which is only 10 times the cost of training a single RSN (8 GPU days).

Visualizing and analyzing LidarNASNet We visualize the macro-level architecture of LidarNASNet in Fig. 3. We start by discussing LidarNASNet-P. At the macro-level, the evolution decided to add a 2D U-Net that enhances the features for each range image pixel before voxelization to the pillar view. This change alone improves the 3D AP from 67.6 to 72.3. The evolution also learned to increase the voxelization granularity from 0.32×0.32 to 0.25×0.25, which is

[3] In fact this architecture was sampled / discovered during our evolution.

a micro-level change that is not reflected in Fig. 3. This change further improves the 3D AP to the 73.2 reported in Tab. 2.

For LidarNASNet-R, notice that though the search was conducted separately for the vehicle and pedestrian class, *the same macro-level architecture design was found*, which is a positive signal regarding the generality of the found design. Specifically, LidarNASNet-R adds a pillar view in the first stage, as well as the associated sparse 2D U-Net. The idea of adding a pillar view resembles MVF [54, 34] (though the sparse format is used here while MVF did not consider sparse operations), making LidarNASNet-R a hybrid between RSN and MVF, two very successful human designs. The voxelization granularity of this sparse 2D U-Net is 0.32×0.32, and the number of channels F is 16. In the vehicle variant, the number of channels for the original perspective view is halved (from 16 to 8). In the pedestrian variant, this is reduced even more aggressively (from 16 to 3).

The search space is challenging A common critique on some of the NAS literature is that the search space can be "easy" in the sense that even random sampling of architectures (and taking the argmax) can find high-quality architectures indistinguishable from those found by NAS [21]. We prove our search space is not trivial, by training 100 architectures randomly generated by the procedure detailed in the supplementary material. Figure 4 shows the side-by-side comparison of these random architectures against our LidarNAS evolution. It is clear that randomly sampled architectures have much worse qualities, in terms of both detection AP and latency. Not only does this illustrate that the search space we consider is challenging and nontrivial, but also justifies our use of warm starting.

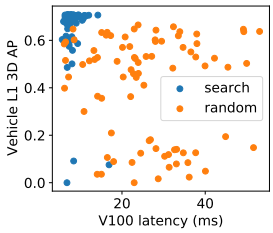


Fig. 4: Randomly sampled architectures (orange) in the LidarNAS search space have worse average and higher variance, calling for warm starting (blue).

Lessons from the sampled architectures In addition to fixating on the top-performing architecture, there are also lessons to be learned in the hundreds of architectures sampled. We now choose a few angles to analyze these data.

Using the LidarNAS evolution data points, we investigate architectures that only mutated the "layer" aspect (i.e. the last two mutation choices in Sec. 4.2) versus the rest. The AP standard deviation of the two subsets are 0.04 and 0.14 respectively, which confirms that on average, mutating "transforms" results in more aggressive changes than mutating "layers" only.

Using the random architectures data points, we study which views and stages have the most direct effect on detection quality. Specifically, we run a linear regression from the 12-dimensional binary feature indicating whether the corresponding branch exists in the architecture to the L1 3D AP. The coefficients are

visualized in Fig. 5. By comparing the columns, it is clear that later stages have a much more direct influence on the detection AP than earlier stages. By comparing the rows of the last column, top-down views (voxel and pillar) positively influence the detection AP, while the perspective view impacts it negatively. This again resonates with the belief that detection from the perspective view tends to be more challenging and requires more specialized treatment.

Using the random architectures data points, we also study the effect of dense vs sparse on latency. Recall that in our LidarNAS framework, the perspective view and the pillar view are the two that allow both dense and sparse formats. For each view, we run a linear regression to latency from a 3-dimensional feature, indicating the total number of empty / dense / sparse branches. For the perspective view, the coefficients are $[-5.24, -1.65, 6.90]$. For the pillar view, the coefficients are $[-3.03, 3.06, -0.03]$. The first coefficient is the most negative for both, which is expected, because the more empty branches you have, the smaller the latency is. Interestingly, the coefficients reveal that on the perspective view, using more sparse branches results in larger latency, whereas on the pillar view, using more sparse branches results in smaller latency. This shows that sparse operations can offer speedup but not always: it depends on whether the view inherently has high sparsity.

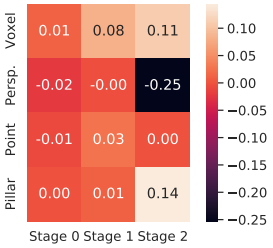


Fig. 5: Linear regression from the presence of individual views and stages to detection AP.

6 Conclusion

This paper aims to achieve two goals for neural architecture research for 3D: first, a unified framework that summarizes and organizes existing designs, and second, an architecture search exploration enabled by this framework. We demonstrate the generality of our LidarNAS framework, not only through pictorial illustration, but also through empirical experiments. Then, we successfully and automatically discovered LidarNASNet, which achieves state-of-the-art results on the Waymo Open Dataset 3D object detection. The searched architecture is interesting: not only is it identical when searching on two different classes, but also embodies and reaffirms shades from existing designs.

There are still many limitations in this work, and we look forward to addressing them in future research. First, while the *transforms* coverage is fairly complete, the *layers* currently implemented do not capture much diversity, and we shall add more powerful layer choices into the search space, such as Transformers [53, 10, 27]. Second, all search experiments are single-frame; in extending to multi-frame, challenges include more memory pressure and the additional complication over which stage to perform temporal fusion.

References

1. Ba, J.L., Kiros, J.R., Hinton, G.E.: Layer normalization. arXiv preprint arXiv:1607.06450 (2016)
2. Bae, W., Lee, S., Lee, Y., Park, B., Chung, M., Jung, K.H.: Resource optimized neural architecture search for 3d medical image segmentation. In: International Conference on Medical Image Computing and Computer-Assisted Intervention. pp. 228–236. Springer (2019)
3. Baker, B., Gupta, O., Naik, N., Raskar, R.: Designing neural network architectures using reinforcement learning. arXiv preprint arXiv:1611.02167 (2016)
4. Bender, G., Liu, H., Chen, B., Chu, G., Cheng, S., Kindermans, P.J., Le, Q.V.: Can weight sharing outperform random architecture search? an investigation with tunas. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 14323–14332 (2020)
5. Bewley, A., Sun, P., Mensink, T., Anguelov, D., Sminchisescu, C.: Range conditioned dilated convolutions for scale invariant 3d object detection. arXiv preprint arXiv:2005.09927 (2020)
6. Chai, Y., Sun, P., Ngiam, J., Wang, W., Caine, B., Vasudevan, V., Zhang, X., Anguelov, D.: To the point: Efficient 3d object detection in the range image with graph convolution kernels. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 16000–16009 (2021)
7. Chen, L.C., Collins, M.D., Zhu, Y., Papandreou, G., Zoph, B., Schroff, F., Adam, H., Shlens, J.: Searching for efficient multi-scale architectures for dense image prediction. arXiv preprint arXiv:1809.04184 (2018)
8. Chen, X., Ma, H., Wan, J., Li, B., Xia, T.: Multi-view 3d object detection network for autonomous driving. In: Proceedings of the IEEE conference on Computer Vision and Pattern Recognition. pp. 1907–1915 (2017)
9. Deng, J., Shi, S., Li, P., Zhou, W., Zhang, Y., Li, H.: Voxel r-cnn: Towards high performance voxel-based 3d object detection. arXiv preprint arXiv:2012.15712 (2020)
10. Engel, N., Belagiannis, V., Dietmayer, K.: Point transformer. IEEE Access **9**, 134826–134840 (2021)
11. Engelcke, M., Rao, D., Wang, D.Z., Tong, C.H., Posner, I.: Vote3deep: Fast object detection in 3d point clouds using efficient convolutional neural networks. In: 2017 IEEE International Conference on Robotics and Automation (ICRA). pp. 1355–1361. IEEE (2017)
12. Fan, L., Xiong, X., Wang, F., Wang, N., Zhang, Z.: Rangedet: In defense of range view for lidar-based 3d object detection. arXiv preprint arXiv:2103.10039 (2021)
13. Geiger, A., Lenz, P., Urtasun, R.: Are we ready for autonomous driving? the kitti vision benchmark suite. In: 2012 IEEE conference on computer vision and pattern recognition. pp. 3354–3361. IEEE (2012)
14. Ghiasi, G., Lin, T.Y., Le, Q.V.: Nas-fpn: Learning scalable feature pyramid architecture for object detection. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 7036–7045 (2019)
15. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
16. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: International conference on machine learning. pp. 448–456. PMLR (2015)

17. Kim, S., Kim, I., Lim, S., Baek, W., Kim, C., Cho, H., Yoon, B., Kim, T.: Scalable neural architecture search for 3d medical image segmentation. In: International Conference on Medical Image Computing and Computer-Assisted Intervention. pp. 220–228. Springer (2019)
18. Lang, A.H., Vora, S., Caesar, H., Zhou, L., Yang, J., Beijbom, O.: Pointpillars: Fast encoders for object detection from point clouds. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 12697–12705 (2019)
19. Li, G., Qian, G., Delgadillo, I.C., Muller, M., Thabet, A., Ghanem, B.: Sgas: Sequential greedy architecture search. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 1620–1630 (2020)
20. Li, G., Xu, M., Giancola, S., Thabet, A., Ghanem, B.: Lc-nas: Latency constrained neural architecture search for point cloud networks. arXiv preprint arXiv:2008.10309 (2020)
21. Li, L., Talwalkar, A.: Random search and reproducibility for neural architecture search. In: Uncertainty in artificial intelligence. pp. 367–377. PMLR (2020)
22. Liu, C., Chen, L.C., Schroff, F., Adam, H., Hua, W., Yuille, A.L., Fei-Fei, L.: Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 82–92 (2019)
23. Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.J., Fei-Fei, L., Yuille, A., Huang, J., Murphy, K.: Progressive neural architecture search. In: Proceedings of the European conference on computer vision (ECCV). pp. 19–34 (2018)
24. Liu, H., Simonyan, K., Yang, Y.: Darts: Differentiable architecture search. arXiv preprint arXiv:1806.09055 (2018)
25. Liu, Z., Tang, H., Lin, Y., Han, S.: Point-voxel cnn for efficient 3d deep learning. arXiv preprint arXiv:1907.03739 (2019)
26. Ma, Z., Zhou, Z., Liu, Y., Lei, Y., Yan, H.: Auto-orvnet: Orientation-boosted volumetric neural architecture search for 3d shape classification. IEEE Access **8**, 12942–12954 (2019)
27. Mao, J., Xue, Y., Niu, M., Bai, H., Feng, J., Liang, X., Xu, H., Xu, C.: Voxel transformer for 3d object detection. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 3164–3173 (2021)
28. Meyer, G.P., Laddha, A., Kee, E., Vallespi-Gonzalez, C., Wellington, C.K.: Lasernet: An efficient probabilistic 3d object detector for autonomous driving. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 12677–12686 (2019)
29. Ngiam, J., Caine, B., Han, W., Yang, B., Chai, Y., Sun, P., Zhou, Y., Yi, X., Alsharif, O., Nguyen, P., et al.: Starnet: Targeted computation for object detection in point clouds. arXiv preprint arXiv:1908.11069 (2019)
30. Pham, H., Guan, M., Zoph, B., Le, Q., Dean, J.: Efficient neural architecture search via parameters sharing. In: International Conference on Machine Learning. pp. 4095–4104. PMLR (2018)
31. Qi, C.R., Litany, O., He, K., Guibas, L.J.: Deep hough voting for 3d object detection in point clouds. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 9277–9286 (2019)
32. Qi, C.R., Su, H., Mo, K., Guibas, L.J.: Pointnet: Deep learning on point sets for 3d classification and segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 652–660 (2017)
33. Qi, C.R., Yi, L., Su, H., Guibas, L.J.: Pointnet++: Deep hierarchical feature learning on point sets in a metric space. arXiv preprint arXiv:1706.02413 (2017)

34. Qi, C.R., Zhou, Y., Najibi, M., Sun, P., Vo, K., Deng, B., Anguelov, D.: Offboard 3d object detection from point cloud sequences. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 6134–6144 (2021)
35. Real, E., Aggarwal, A., Huang, Y., Le, Q.V.: Regularized evolution for image classifier architecture search. In: Proceedings of the aaai conference on artificial intelligence. vol. 33, pp. 4780–4789 (2019)
36. Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y.L., Tan, J., Le, Q.V., Kurakin, A.: Large-scale evolution of image classifiers. In: International Conference on Machine Learning. pp. 2902–2911. PMLR (2017)
37. Ronneberger, O., Fischer, P., Brox, T.: U-net: Convolutional networks for biomedical image segmentation. In: International Conference on Medical image computing and computer-assisted intervention. pp. 234–241. Springer (2015)
38. Shi, S., Guo, C., Jiang, L., Wang, Z., Shi, J., Wang, X., Li, H.: Pv-rcnn: Point-voxel feature set abstraction for 3d object detection. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 10529–10538 (2020)
39. Shi, S., Guo, C., Yang, J., Li, H.: Pv-rcnn: The top-performing lidar-only solutions for 3d detection/3d tracking/domain adaptation of waymo open dataset challenges. arXiv preprint arXiv:2008.12599 (2020)
40. So, D., Le, Q., Liang, C.: The evolved transformer. In: International Conference on Machine Learning. pp. 5877–5886. PMLR (2019)
41. So, D.R., Mańke, W., Liu, H., Dai, Z., Shazeer, N., Le, Q.V.: Primer: Searching for efficient transformers for language modeling. arXiv preprint arXiv:2109.08668 (2021)
42. Sun, P., Kretzschmar, H., Dotiwalla, X., Chouard, A., Patnaik, V., Tsui, P., Guo, J., Zhou, Y., Chai, Y., Caine, B., et al.: Scalability in perception for autonomous driving: Waymo open dataset. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 2446–2454 (2020)
43. Sun, P., Wang, W., Chai, Y., Elsayed, G., Bewley, A., Zhang, X., Sminchisescu, C., Anguelov, D.: Rsn: Range sparse net for efficient, accurate lidar 3d object detection. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 5725–5734 (2021)
44. Tan, M., Le, Q.: Efficientnet: Rethinking model scaling for convolutional neural networks. In: International Conference on Machine Learning. pp. 6105–6114. PMLR (2019)
45. Tang, H., Liu, Z., Zhao, S., Lin, Y., Lin, J., Wang, H., Han, S.: Searching efficient 3d architectures with sparse point-voxel convolution. In: European Conference on Computer Vision. pp. 685–702. Springer (2020)
46. Wang, Y., Fathi, A., Kundu, A., Ross, D.A., Pantofaru, C., Funkhouser, T., Solomon, J.: Pillar-based object detection for autonomous driving. In: Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXII 16. pp. 18–34. Springer (2020)
47. Wong, K.C., Moradi, M.: Segnas3d: Network architecture search with derivative-free global optimization for 3d image segmentation. In: International Conference on Medical Image Computing and Computer-Assisted Intervention. pp. 393–401. Springer (2019)
48. Xu, H., Yao, L., Zhang, W., Liang, X., Li, Z.: Auto-fpn: Automatic network architecture adaptation for object detection beyond classification. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 6649–6658 (2019)
49. Yan, Y., Mao, Y., Li, B.: Second: Sparsely embedded convolutional detection. Sensors **18**(10), 3337 (2018)

50. Yang, B., Luo, W., Urtasun, R.: Pixor: Real-time 3d object detection from point clouds. In: Proceedings of the IEEE conference on Computer Vision and Pattern Recognition. pp. 7652–7660 (2018)
51. Yin, T., Zhou, X., Krahenbuhl, P.: Center-based 3d object detection and tracking. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 11784–11793 (2021)
52. Yu, Q., Yang, D., Roth, H., Bai, Y., Zhang, Y., Yuille, A.L., Xu, D.: C2fnas: Coarse-to-fine neural architecture search for 3d medical image segmentation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 4126–4135 (2020)
53. Zhao, H., Jiang, L., Jia, J., Torr, P.H., Koltun, V.: Point transformer. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 16259–16268 (2021)
54. Zhou, Y., Sun, P., Zhang, Y., Anguelov, D., Gao, J., Ouyang, T., Guo, J., Ngiam, J., Vasudevan, V.: End-to-end multi-view fusion for 3d object detection in lidar point clouds. In: Conference on Robot Learning. pp. 923–932. PMLR (2020)
55. Zhou, Y., Tuzel, O.: Voxelnet: End-to-end learning for point cloud based 3d object detection. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 4490–4499 (2018)
56. Zhu, Z., Liu, C., Yang, D., Yuille, A., Xu, D.: V-nas: Neural architecture search for volumetric medical image segmentation. In: 2019 International Conference on 3D Vision (3DV). pp. 240–248. IEEE (2019)
57. Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning. arXiv preprint arXiv:1611.01578 (2016)