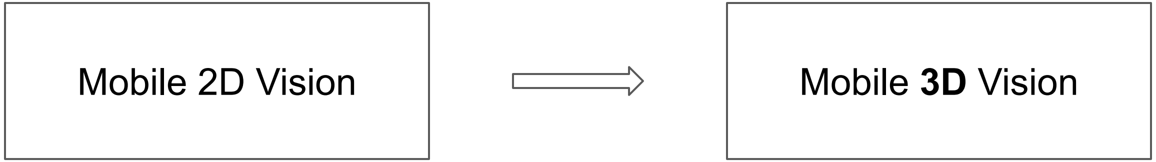
Toward Mobile 3D Vision

Huanle Zhang[#], Bo Han[&], Prasant Mohapatra[#]

[#]University of California, Davis
Davis, California, USA

[&]AT&T Labs - Research
Bedminster, New Jersey, USA

Position Paper



Mobile 2D Vision

Mobile **3D** Vision

Challenges: (1) **computation** intensive; (2) **memory** hungry

Research Agenda: potential research areas for improving the efficiency of executing 3D vision in real-time on mobile device

3D Vision is Essential

3D vs. 2D: **depth** information, crucial for many applications

(b) Autonomous driving

(c) Drone

(d) Co-present avatar

Image sources: www.vectorstock.com; www.store.dji.com;
https://channels.theinnovationenterprise.com/articles/new-virtual-avatar-star-to-bring-books-to-life-through-sign-language

Key Components for 3D Vision

1. Object Detection

Each 3D object of interest is localized

2. Scene Segmentation

Each input point is classified with a label

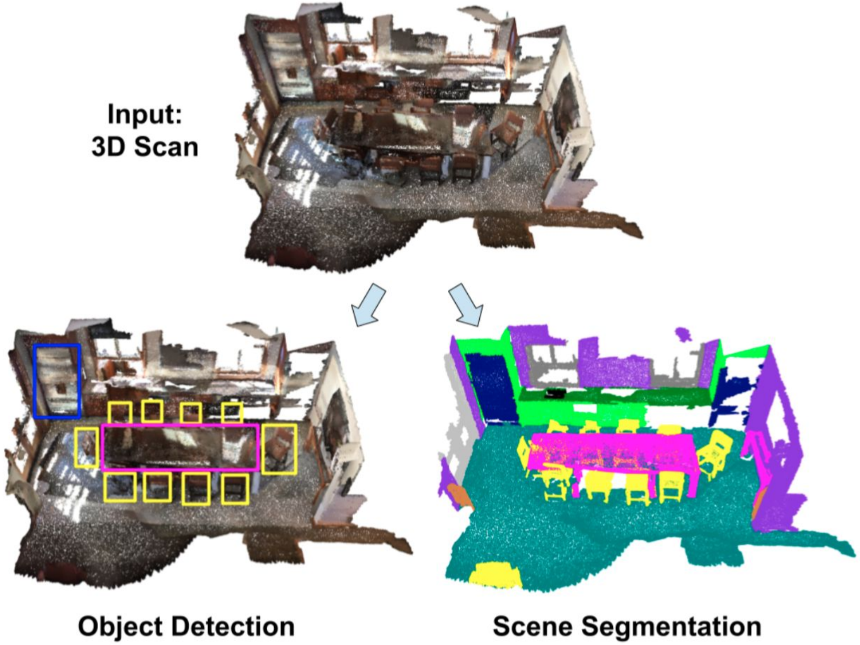


Illustration of 3D object detection and scene segmentation

3D Data Representation

1. 3D Mesh

   Not DNN-friendly

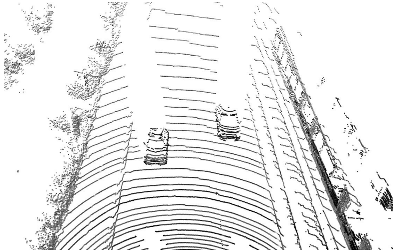
(a) A 3D mesh of cat[1]

2. Point Cloud: an unordered set of points.
   Each point in (X, Y, Z, P) where P is a property

   E.g.,

   P = ∅ in the ShapeNet dataset[2]
   P = I (reflectance value) in the KITTI dataset[3]
   P = (R, G, B) in the ScanNet dataset[4]

(b) A (X, Y, Z) point cloud



(c) A (X, Y, Z, I) point cloud

(d) A (X, Y, Z, R, G, B) point cloud

[1] Image sources: https://www.pinterest.com/pin/325244404324563579/
[2] ShapeNet dataset: https://www.shapenet.org/
[3] KITTI dataset: http://www.cvlibs.net/datasets/kitti/
[4] ScanNet dataset: http://www.scan-net.org/

Feature Extraction From Point Cloud

Different methods of feature extraction result in different degrees of data dimensionality, which in turn determines the DNN model complexity

1. Converting to 2D Feature Vectors

   E.g., ComplexYolo [1]

2. A Feature Vector for Each Grid Cell

   E.g., VoxelNet [2]

3. A Feature Vector for Each Pillar

   E.g., PointPillars [3]

4. A Feature Vector for Each Point

   E.g., SparseConvNet [4]

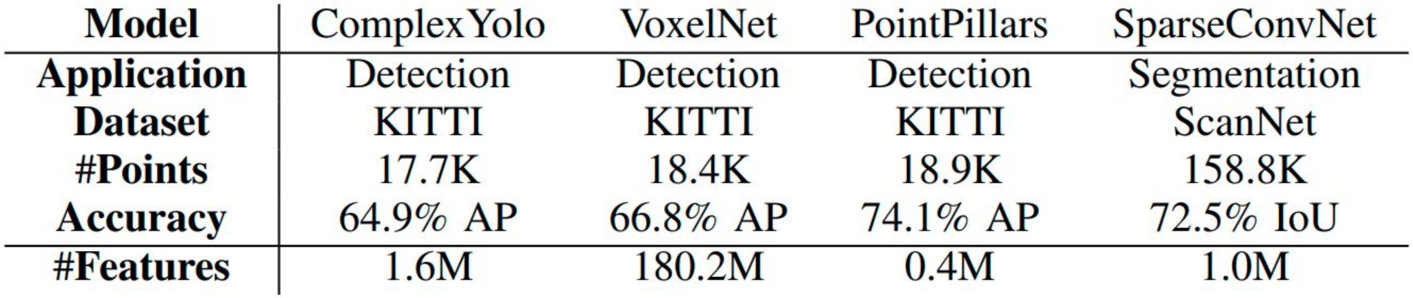
[1] Martin Simon, Stefan Milz, Karl Amende, and Horst-Michael Gross. Complex-YOLO: An Euler-Region-Proposal for Real-time 3D Object Detection on Point Clouds. In Proceedings of European Conference on Computer Vision Workshop (ECCV Workshop), 2018.

[2] Yin Zhou and Oncel Tuzel. VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection. In Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2018.

[3] Alex H. Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. PointPillars: Fast Encoders for Object Detection From Point Clouds. In Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019.

[4] Benjamin Graham, Martin Engelcke, and Laurens van der Maaten. 3D Semantic Segmentation with Submanifold Sparse Convolutional Networks. In Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2018

Comparison of Selected DNN Models



Model	ComplexYolo	VoxelNet	PointPillars	SparseConvNet
Application	Detection	Detection	Detection	Segmentation
Dataset	KITTI	KITTI	KITTI	ScanNet
#Points	17.7K	18.4K	18.9K	158.8K
Accuracy	64.9% AP	66.8% AP	74.1% AP	72.5% IoU
#Features	1.6M	180.2M	0.4M	1.0M

During inference, the models make predictions based on different number of input features

Measurement Setup

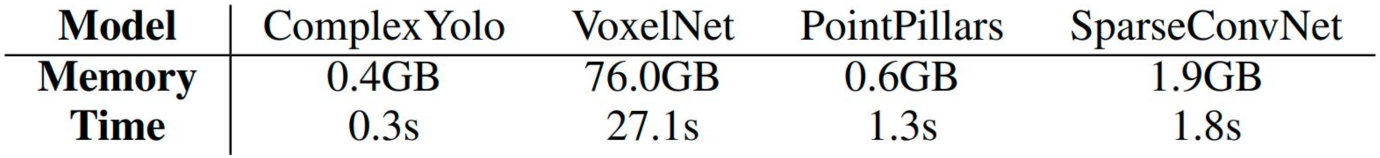
Hardware:

- A Server (Dell PowerEdge T640 with 40 2.2GHz CPU cores)
- Phones (Huawei Mate 20 and Google Pixel 2)

Tensorflow/Tensorflow Lite for ComplexYolo, VoxelNet and PointPillars.

PyTorch for SparseConvNet

Running Models on Server



Model	ComplexYolo	VoxelNet	PointPillars	SparseConvNet
Memory	0.4GB	76.0GB	0.6GB	1.9GB
Time	0.3s	27.1s	1.3s	1.8s

Memory usage and execution time of selected DNN models on a commodity server

Performance difference: 90X in speed and 190X in memory

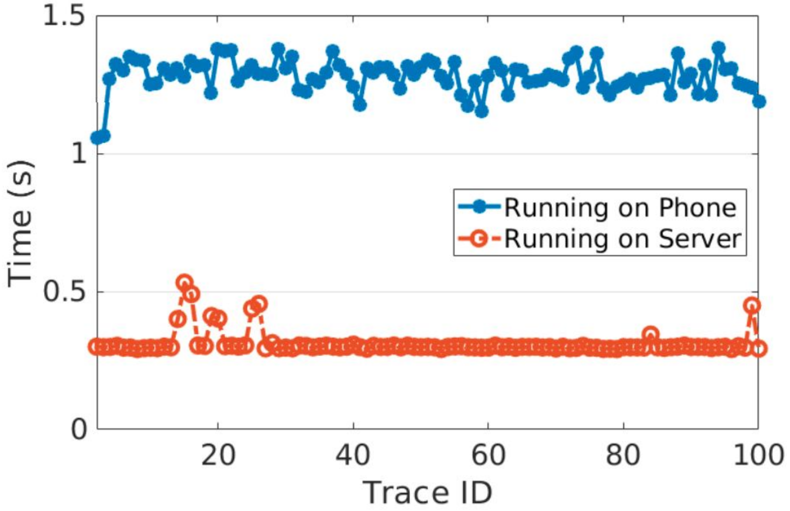
In addition:

(1) ComplexYolo is lightweight
(2) VoxelNet is extremely slow
(3) PointPillars dramatically reduces the overheads compared to VoxelNet
(4) SparseConvNet is efficient

Phone vs. Server (Tensorflow Lite Compatible)

ComplexYolo, 100 runs

Huawei Mate 20 takes 1.3 seconds per point cloud, 3.9 times slower than the server



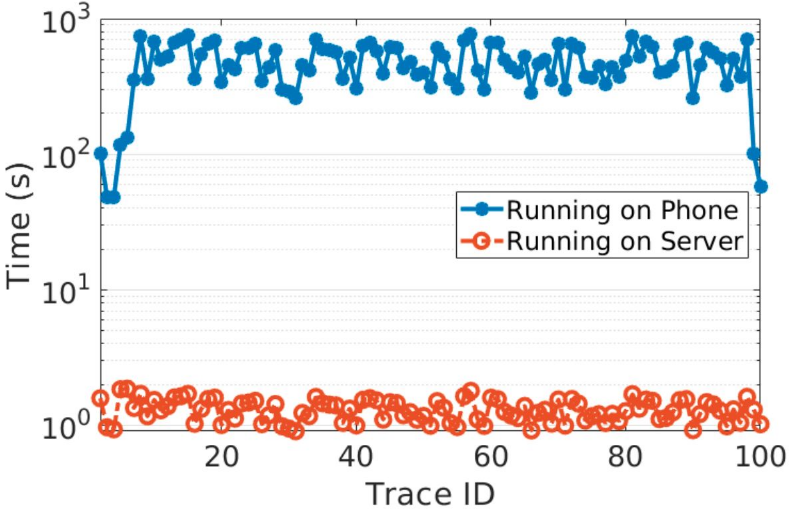
ComplexYolo

Phone Versus Server (Tensorflow Lite Incompatible)

PointPillars, 100 runs

Variable-length 1D convolutional layer is not supported by Tensorflow Lite

Huawei Mate 20 runs 375.5 times slower than the server



PointPillars

Phone GPU versus CPU

Using GPU may be slower than CPU if some model operators are not supported by the GPU[1].

Take ComplexYolo as an example:

Phone	CPU only	CPU + GPU
Huawei Mate 20	1.3 seconds	2.3 seconds
Google Pixel 2	2.6 seconds	3.4 seconds

[1] Tensorflow Lite non-supported models and ops of GPU results in performance slower than running on CPU alone. https://www. tensorflow.org/lite/performance/gpu

Experiment Summary

It is challenging to support 3D vision in real time on mobile devices

- Slower than 1 point cloud per second. A continuous vision system requires at least a dozen hertz.
- Larger than 0.4GB memory consumption, which is demanding for smartphones since memory is shared by many applications

Research Agenda

Possible solutions to accelerate 3D vision on mobile devices

- Down-sampling Input
- Offloading
- Model Selection
- Locality in Continuous Vision
- Hardware Parallelism

Proposal 1: Down-sampling Input

Down-sample input so that a more lightweight DNN model can be used

For example, AdaScale [1] trains several 2D object detection models for different image resolutions, and designs a neural network to predict the optimal down-sampling factor for each given image

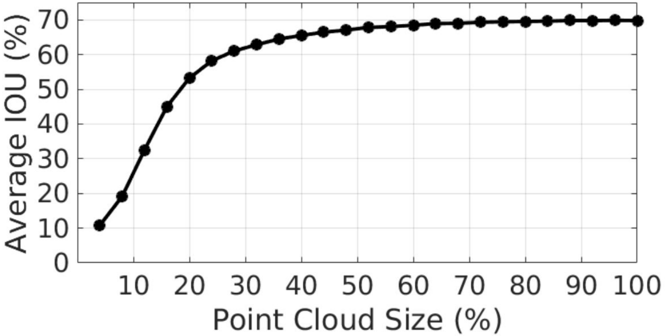
[1]. Ting-Wu Chin, Ruizhou Ding, and Diana Marculescu. AdaScale: Towards Real-time Video Object Detection using Adaptive Scaling. In Proceedings of Conference on Systems and Machine Learning (SysML), 2019.

Proposal 1: Down-sampling Input (Continued)

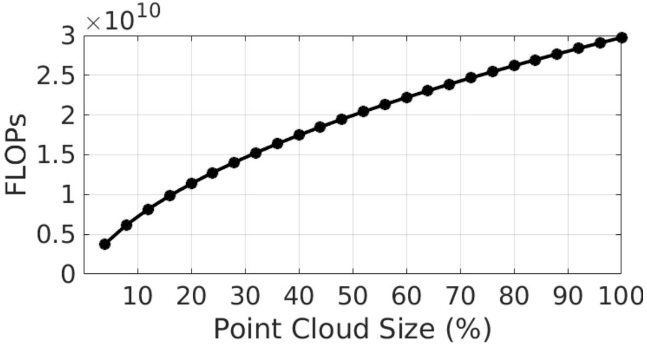
We found that we can use a single pre-trained model for point clouds of any size

1. Accuray remains the same when the input point cloud is sparsified by 40%
2. A point cloud of 50% points takes about ⅔ FLOPs

Challenge: it is unknown how to predict the optimal down-sampling factor for each point cloud



(a) Accuracy



(b) Computation Overhead

Proposal 2: Offloading

Offloading computation-intensive tasks to the cloud can alleviate hardware constraints of mobile device

Offloading schemes:

1. Intermediate Result Offloading, e.g., VisualPrint [1]
2. Partial Raw Data Offloading, e.g., [2]

Challenges

1. Identify Region of Interest (RoI) for point clouds
2. Tradeoff between the pre-processing of raw data and end-to-end latency

[1]. Puneet Jain, Justin Manweiler, and Romit Roy Choudhury. Low Bandwidth Offload for Mobile AR. In Proceedings of International Conference on Emerging Networking Experiments and Technologies (CoNEXT), 2016

[2]. Luyang Liu, Hongyu Li, and Marco Gruteser. Edge Assisted Real-time Object Detection for Mobile Augmented Reality. In Proceedings of ACM International Conference on Mobile Computing and Networking (MobiCom), 2019.

Proposal 3: Model Selection

Select appropriate DNN model according to the run-time resources of mobile devices

Cameras output images of the same resolution, and the models’ computation and memory overhead can be determined in advance to facilitate the selection

Challenge: A 3D scanner generates point clouds with different number of points, e.g., higher point density for furniture than walls

Proposal 4: Locality in Continuous Vision

Object detection is only performed for two frames that are dramatically different and caching is used for frames in between.

For example

1. Tracking image blocks [1]
2. Neural network for object tracking [2]
3. Point cloud tracking, e.g., FlowNet3D [3]

Challenge: a lightweight tracker for point clouds

[1]. Mengwei Xu, Mengze Zhu, Yunxin Liu, Felix Xiaozhu Lin, and Xuanzhe Liu. DeepCache: Principled Cache for Mobile Deep Vision. In Proceedings of ACM International Conference on Mobile Computing and Networking (MobiCom), 2018

[2]. Huizi Mao, Taeyoung Kong, and William J. Dally. CaTDet: Cascaded Tracked Detector for Efficient Object Detection from Video. In Proceedings of Conference on Systems and Machine Learning (SysML), 2019.

[3]. Xingyu Liu, Charles R. Qi, and Leonidas J. Guibas. FlowNet3D: Learning Scene Flow in 3D Point Clouds. In Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019

Proposal 5: Hardware Parallelism

It can greatly speed up model execution if all the resources, e.g, CPU, GPU and DSP on smartphones, can be used in parallel.

Parallelizing DNN based systems

1. Parallelizing DNN Model
2. Parallelizing Input Data, e.g., MobiSR [1]

Challenge: (1) minimize the extra inter-hardware communication overheads; (2) partitioning a point cloud and decides which patch of a point cloud runs in which hardware

[1]. Royson Lee, Stylianos I. Venieris, Lukasz Dudziak, Sourav Bhattacharya, and Nicholas D. Lane. MobiSR: Efficient On-Device SuperResolution through Heterogeneous Mobile Processors. In Proceedings of ACM International Conference on Mobile Computing and Networking (MobiCom), 2019.

Conclusion

Our preliminary measurement study reveals that it is not only computation intensive, but also memory-inefficient for mobile devices to execute existing DNN models for 3D vision directly.

We present a research agenda for accelerating these DNN models and point out several possible solutions to better support continuous 3D vision on mobile devices, by considering the unique characteristics of point clouds.

Thanks

Questions and Answers

Method 1: Converting to 2D Feature Vectors

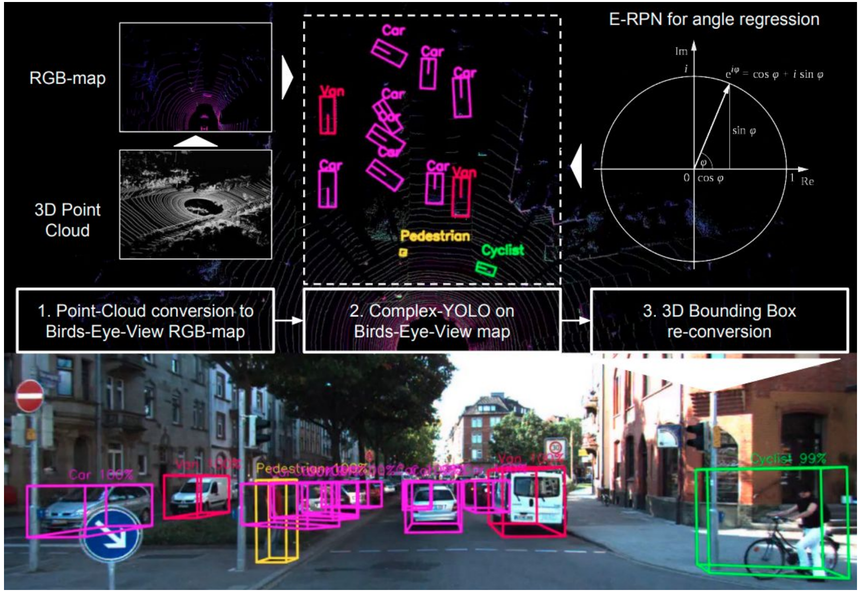
A point cloud is represented by images generated from different viewpoints and viewing angles, and then a 2D DNN model is applied .

A representative DNN model: **ComplexYolo** (ECCV Workshop, 2018)

Convert to a 1024×512 RGB image

A feature vector of 3 for each pixel

Total: **1.6M** features


E-RPN for angle regression
RGB-map
3D Point Cloud
Car
Van
Pedestrian
Cyclist
Im
Re
$e^{i\varphi} = \cos\varphi + i\sin\varphi$
$\sin\varphi$
$\cos\varphi$
φ
0
1
1. Point-Cloud conversion to Birds-Eye-View RGB-map
2. Complex-YOLO on Birds-Eye-View map
3. 3D Bounding Box re-conversion
Cyclist 99%

ComplexYolo structure (image from the original paper)

Method 2: A Feature Vector for Each Grid Cell

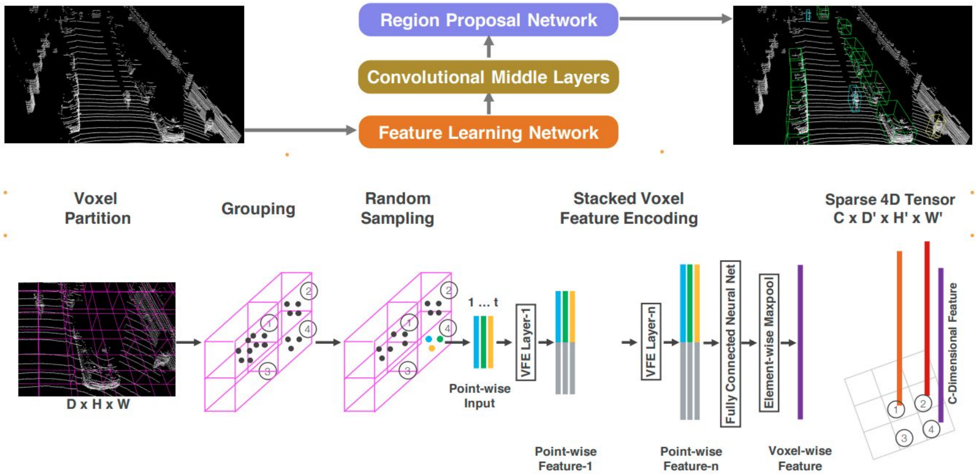
A point cloud is voxelized/partitioned into grid cells and then a feature vector is generated for each cell

A representative DNN model: **VoxelNet** (CVPR, 2018)

Convert to 10×400×352 grid cells

A feature vector of 128 for each cell

Total: **180.2M** features



VoxelNet structure (image from the original paper)

Method 3: A Feature Vector for Each Pillar

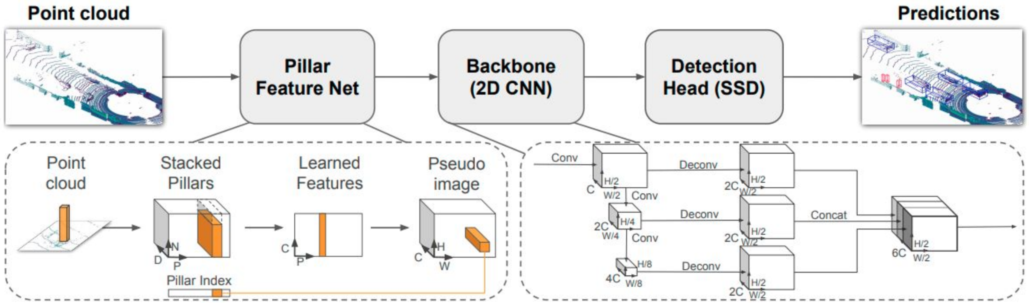
A point cloud is partitioned into pillars and then generates feature vectors for only non-empty pillars that have points

A representative DNN model: **PointPillars** (CVPR, 2019)

Convert to ~5719 non-empty pillars

A feature vector of 64 for each pillar

Total: **~0.4M** features



PointPillars structure (image from the original paper)

Method 4: A Feature Vector for Each Point

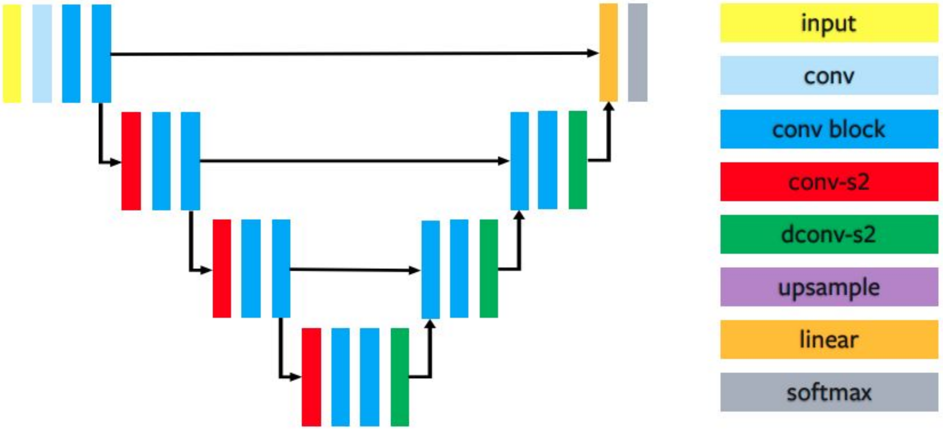
Directly consume points and thus avoid voxelizing/partitioning

A representative DNN model: **SparseConvNet** (CVPR, 2018)

~158.8K points for the ScanNet point cloud

A feature vector of 6 for each point

Total: **~1.0M** features



SparseConvNet structure (image from the original paper)

Comparison of Different Feature Extraction

Method	**Pros**	**Cons**
Converting to 2D Feature Vectors	Widely available 2D models; small computation and memory overheads	Low accuracy; not suitable for 3D semantic segmentation
A Feature Vector for Each Grid Cell	High flexibility for different accuracy targets	Extremely large computation and memory overheads
A Feature Vector for Each Pillar	Reduced computation and memory overheads	Worsened data granularity; does not generalize well to object layouts
A Feature Vector for Each Point	Efficient with regards to computation and memory for sparse point clouds	Not suitable for dense point cloud; no processing of the input data