Institut für Kartographie und Geoinformatik | Leibniz Universität Hannover

Deep Learning II

Claus.Brenner@ikg.uni-hannover.de

Leibniz Universität Hannover

Deep learning applications

- Speech recognition, natural language processing (translation), recommender systems, handwriting recognition, fraud detection, playing games, …
- Images
  - Image recognition (classes: cat, dog, …)
  - Face recognition (log in, surveillance)
  - Medical image analysis (suspicious tissue)
  - Handwriting recognition (ZIP codes, checks)
  - Image description (captions)
  - Semantic segmentation (pixel-wise labels)
  - Object detection and tracking (bounding boxes)
- Convolutional neural networks esp. successful for image data
  - Easy to apply to 2D raster with a few channels (e.g., R, G, B)

Image recognition

Image source: Karen Zack, Twitter

Semantic segmentation

Image source: cityscapes-dataset.com

Object detection

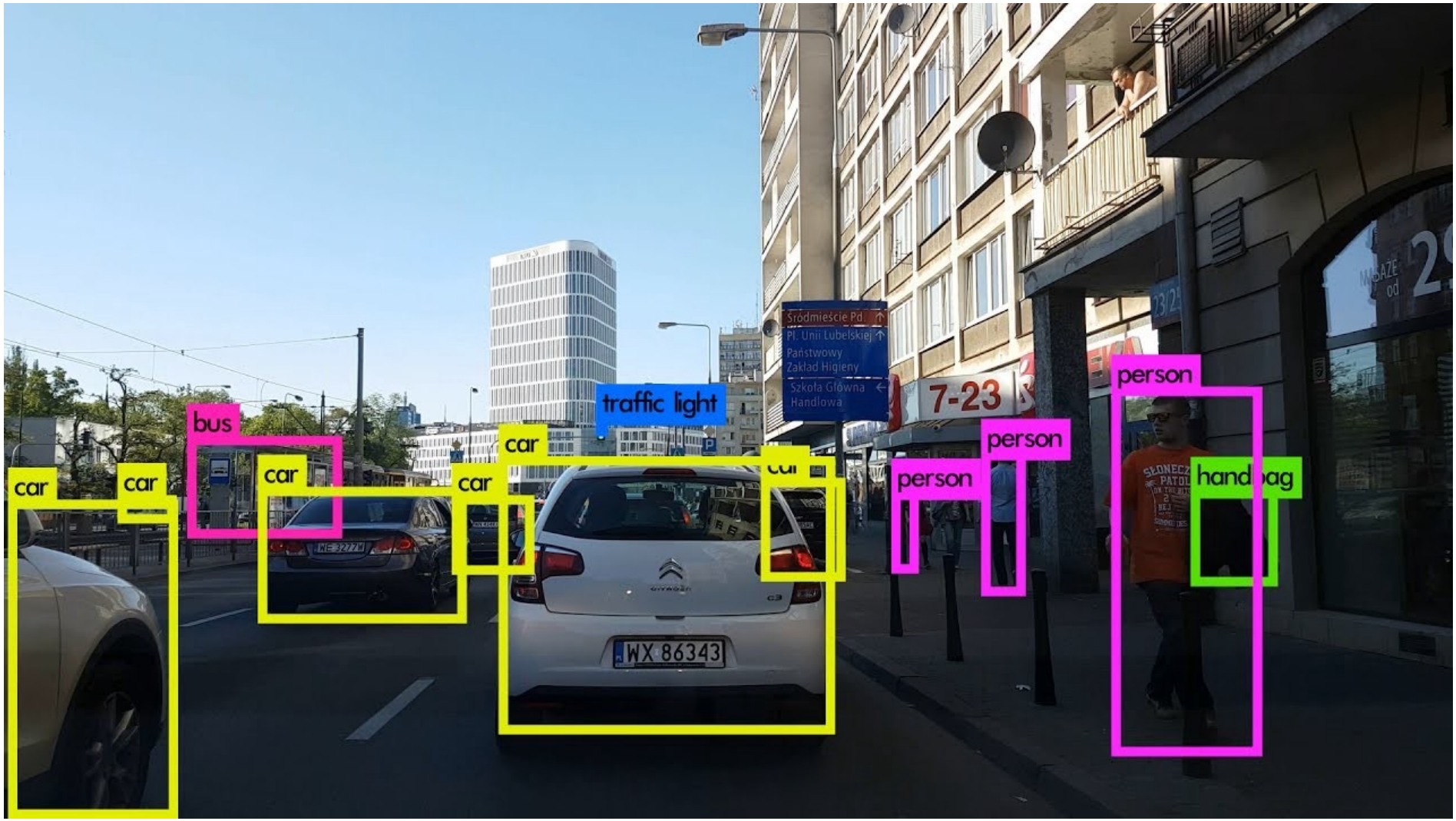


Image source: towardsdatascience.com

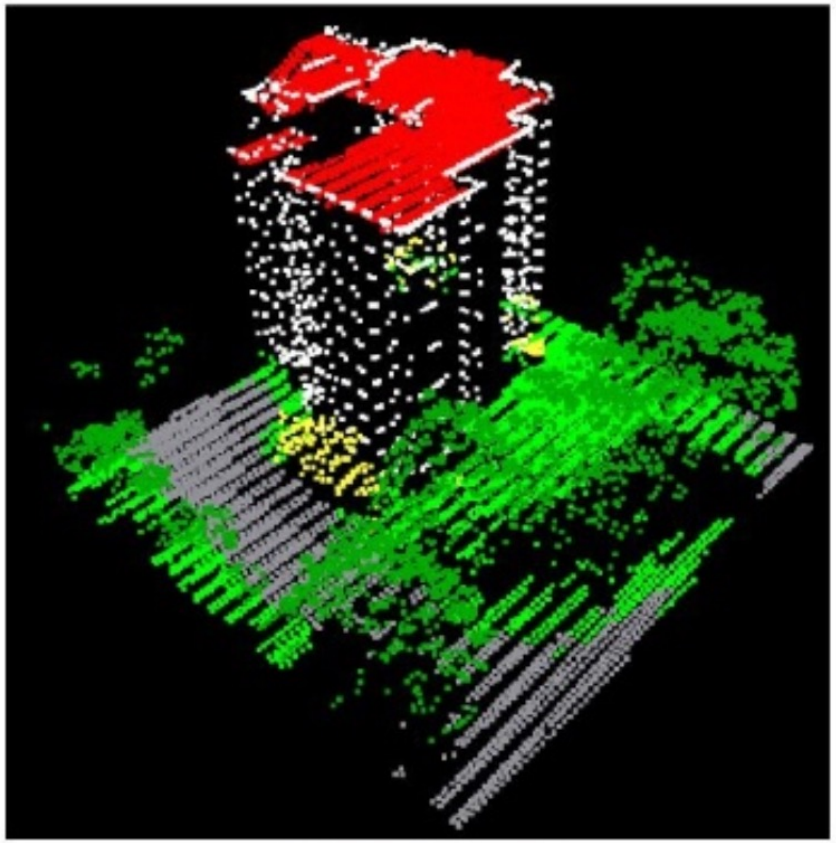
Deep learning for point clouds

Deep learning for point clouds

- Principal problem: point clouds
  - … are **unordered**: no regular raster (as in the case of 2D images)
  - … have **largely varying densities** (usually very dense close to the vehicle, quickly getting sparse with increasing distance, remember the $\propto 1/\cos^2\theta$ rule)

- Basic approaches
  - Sort points into 3D raster (voxels), use 3D convolutions on voxel grid
  - Use projections 3D→2D, then use 2D convolutions on 2D grid
    - Project to (virtual) image plane, project to ground ('BEV', Bird's-eye view)
    - Use 2D polar image of scanner (depth image)
  - Use of unordered point list

Example: Voxel grid



Point Cloud

Voxel Cloud

 Image source: ifp.uni-stuttgart.de/en/research/remote_sensing/als_point_cloud_classification/

Projection to image

Projected point cloud

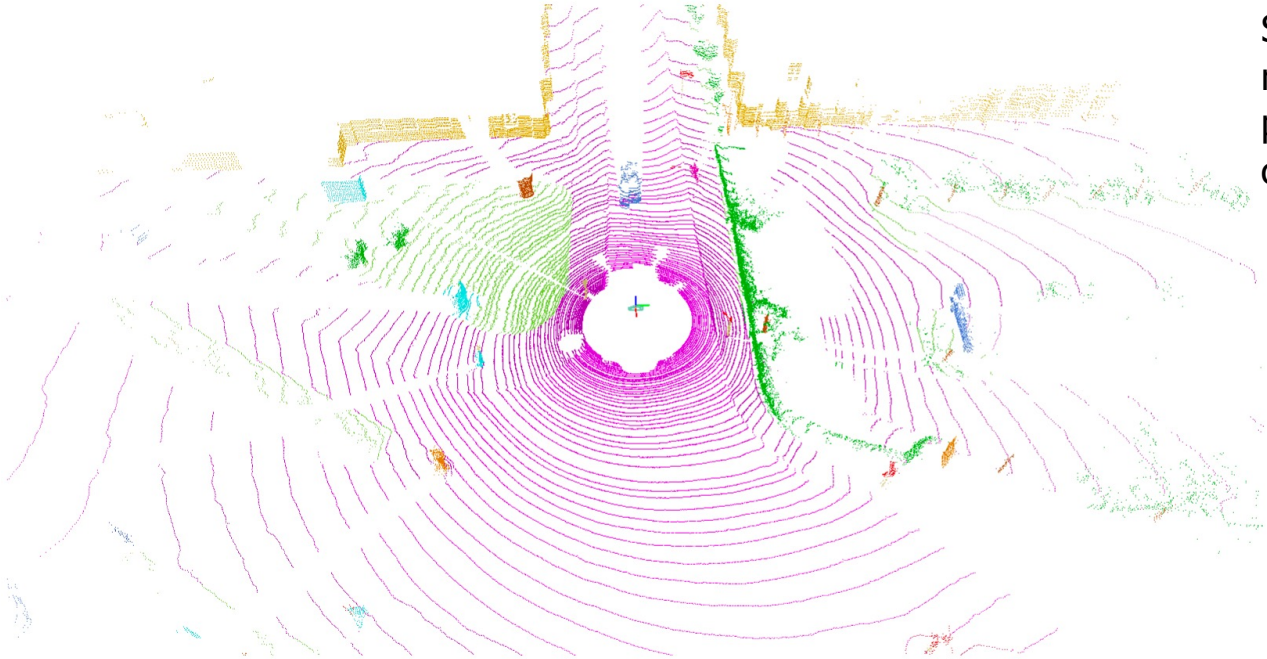
Real image

Generated image

Work by Torben Peters, IKG

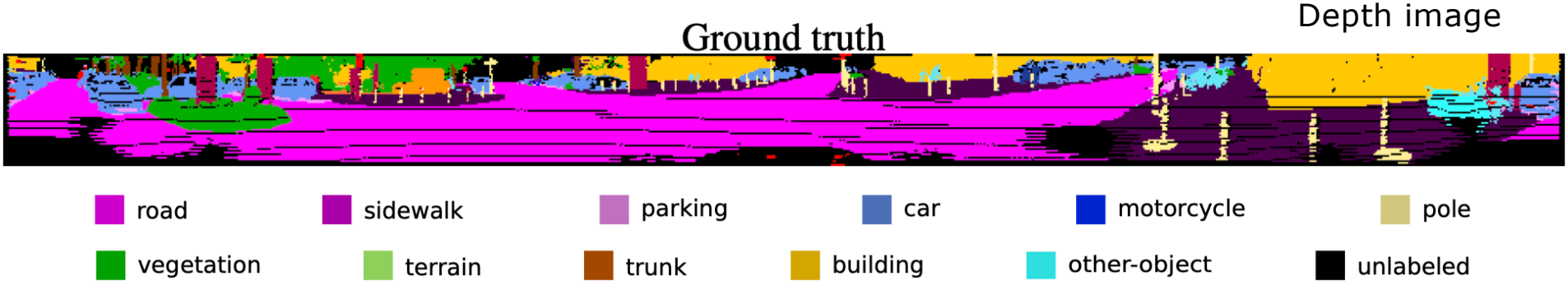
Image source: Peters, T., Brenner, C. Conditional Adversarial Networks for Multimodal Photo-Realistic Point Cloud Rendering. PFG 88, 257–269 (2020). https://doi.org/10.1007/s41064-020-00114-z

Projection: depth image (one rotation of multi-beam rotational scanner)



Single rotation point cloud

Depth image



Images source: Jens Behley, Martin Garbade, Andres Milioto, Jan Quenzel, Sven Behnke, Cyrill Stachniss, Juergen Gall: SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences, arXiv:1904.01416

PointNet

Is based on an unordered point list for classification, part segmentation, or semantic segmentation

PointNet

- CVPR 2017, see https://arxiv.org/abs/1612.00593
- Interesting approach, but nowadays outdated
  - Later: PointNet++, KPConv
- Basic idea:
  - No voxels, no neighbourhoods for spatial convolutions
  - Works on **unordered** $n \times 3$ point list (or $n \times 6$, including normal vectors)
  - To achieve independence from neighbours, each point is processed **individually and identically**
  - This can be achieved with convolutions of kernel size 1
    - E.g. `torch.nn.Conv1d(3, 64, 1)`: from 3 channels (x, y, z) to 64 channels, with kernel size 1.
  - To achieve **independence from ordering**, a symmetric function is used: maximum over columns

PointNet: two variants (in original paper)

- The basic network does a classification
  - The entire point cloud is assigned k output scores (k being number of classes)
- An extension, *segmentation network*, allows to do semantic segmentation and outputs m scores for each of the n points
  - m is the part (in part segmentation) or class (in semantic segmentation)

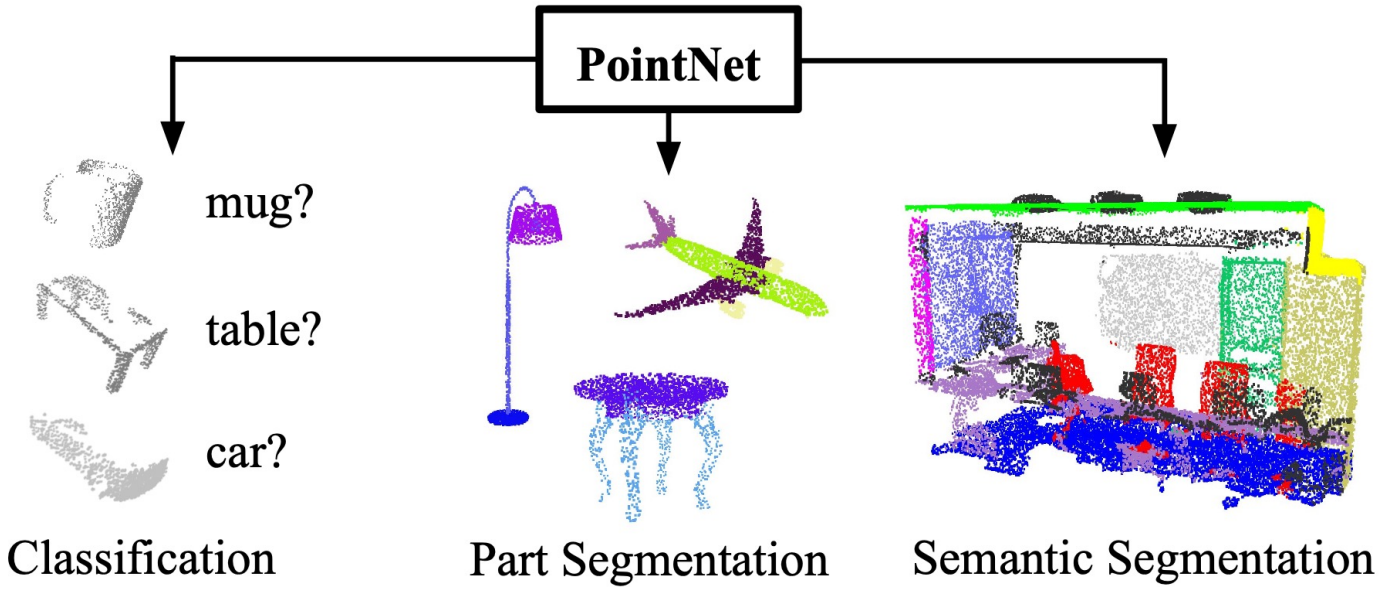


 Image source: original publication, arXiv 1612.00593

PointNet

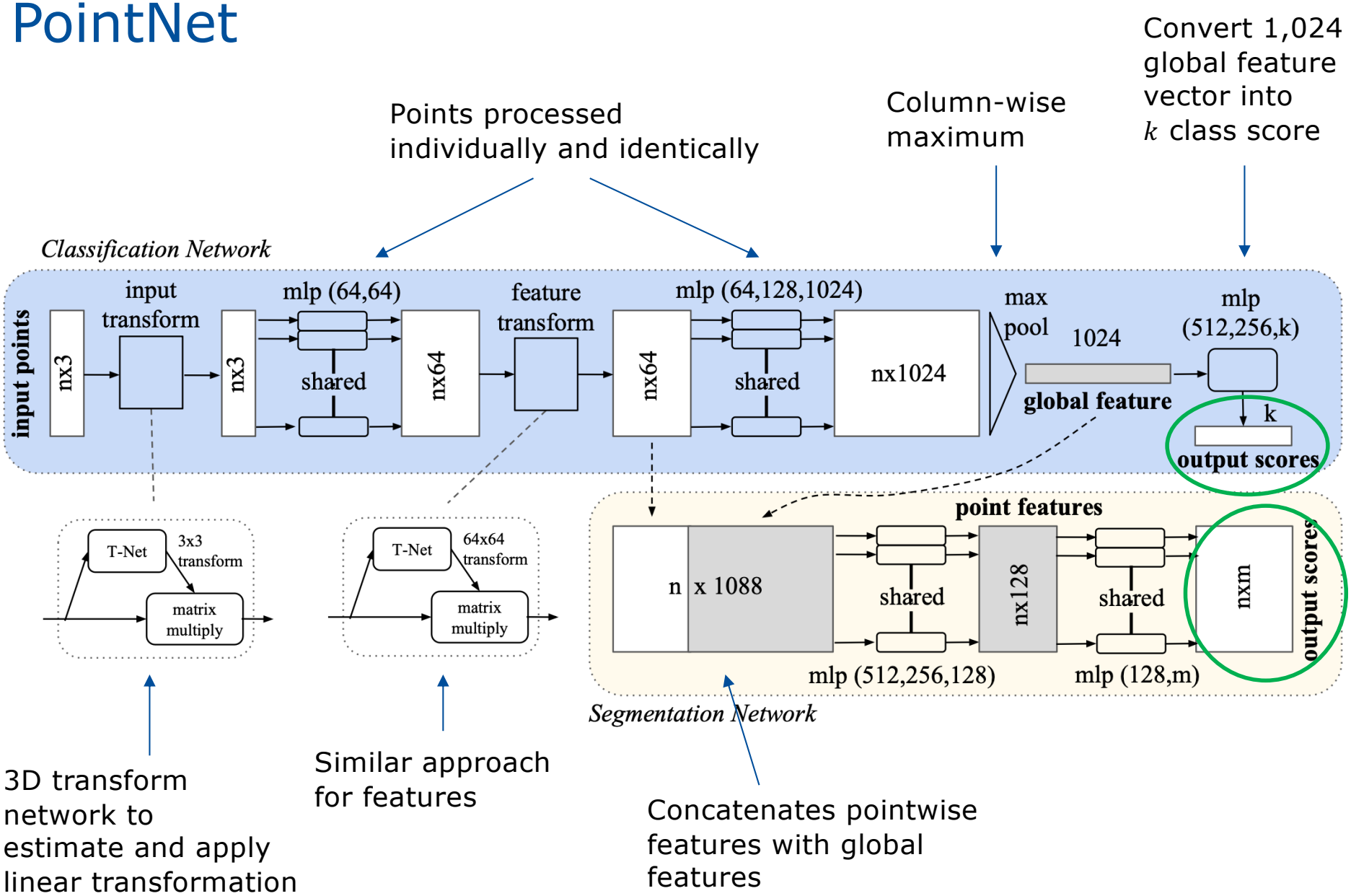


 Image source: original publication, arXiv 1612.00593

Part segmentation results

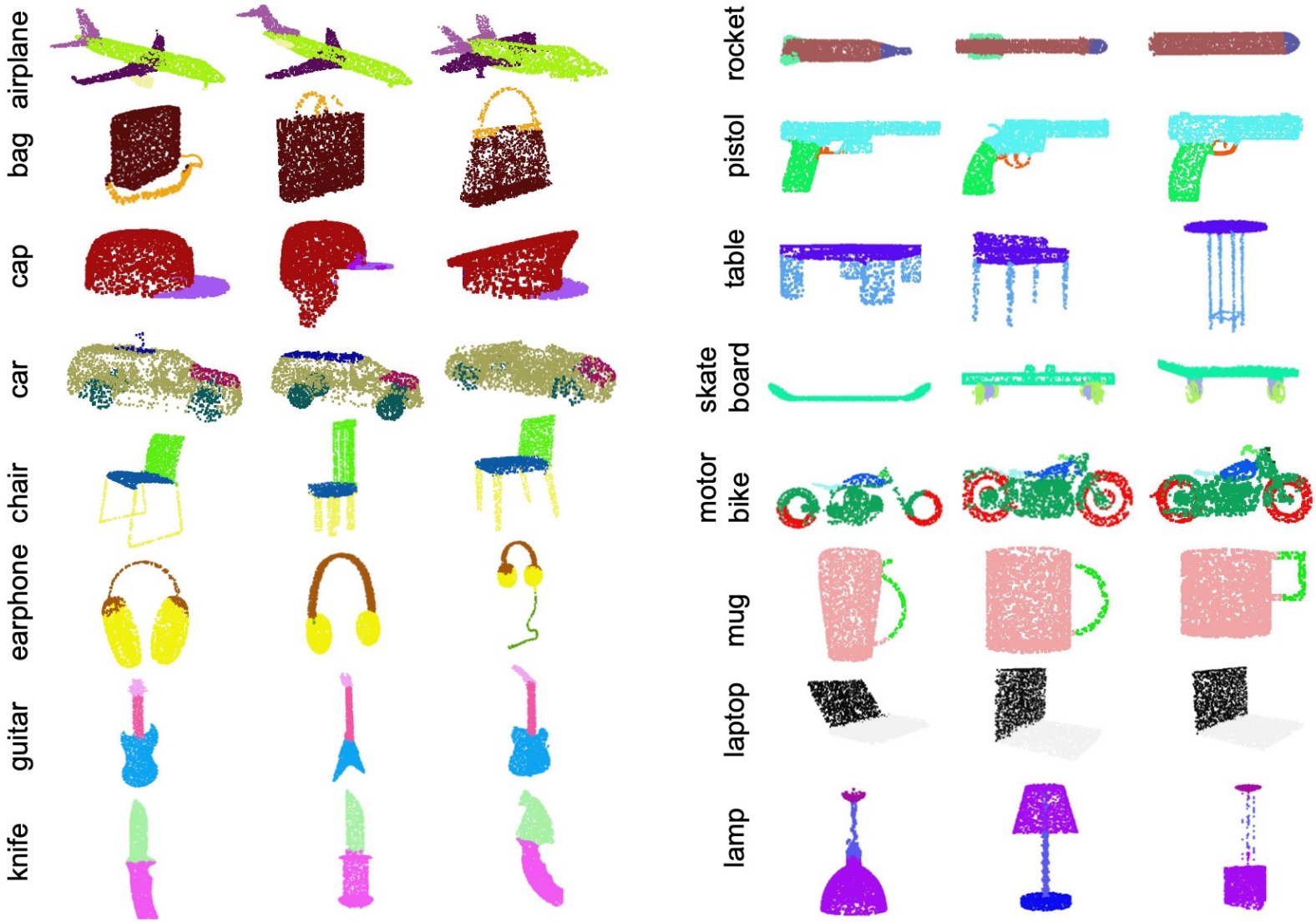


Figure 21. **PointNet segmentation results on complete CAD models.**

 Image source: original publication, arXiv 1612.00593

Semantic segmentation results

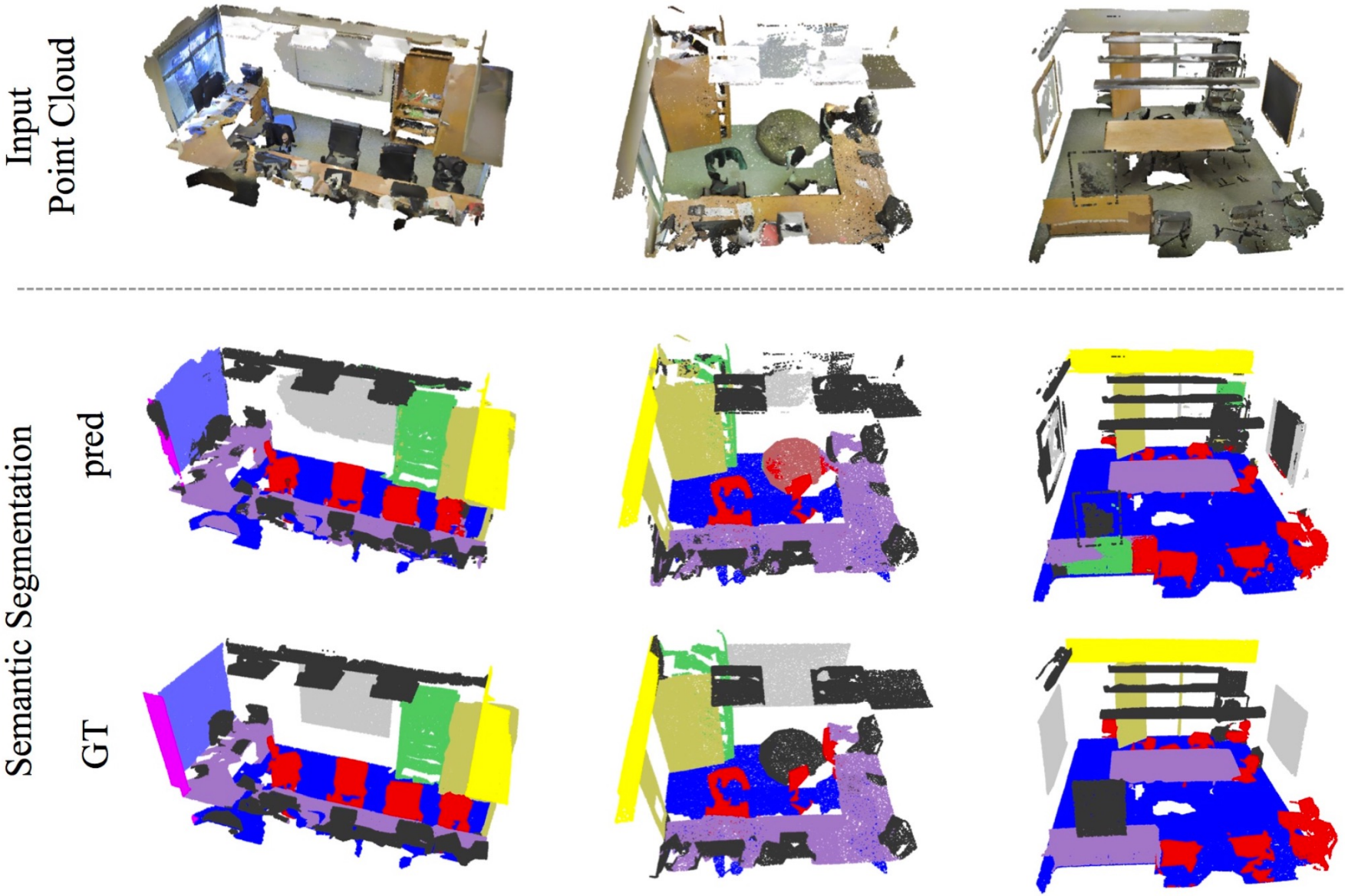


 Image source: original publication, arXiv 1612.00593

KPConv

Uses kernel point convolutions to build multi-layered networks similar to CNN

KPConv

- ICCV 2019, see https://arxiv.org/abs/1904.08889
- In a voxel grid, 'points' are localized by indices
- In a point cloud, they are localized by coordinates
- Usual approach:
  - Project point cloud onto a regular (grid) structure (2D or 3D)
  - Perform convolution on this grid structure (using indices)
- KPConv: Kernel Point Convolution
  - Instead of kernel pixels (grid) use kernel points to define the area where each kernel weight is applied
  - Kernel weights carried by points
  - Area of influence defined by a correlation function

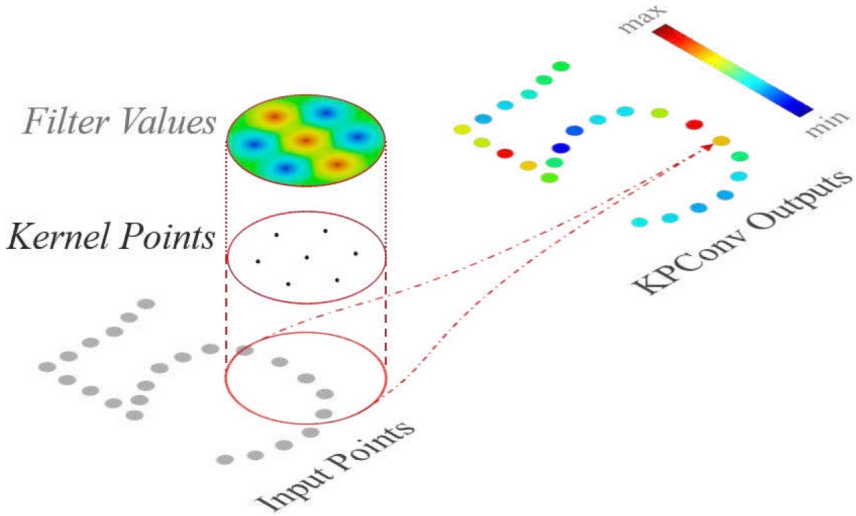
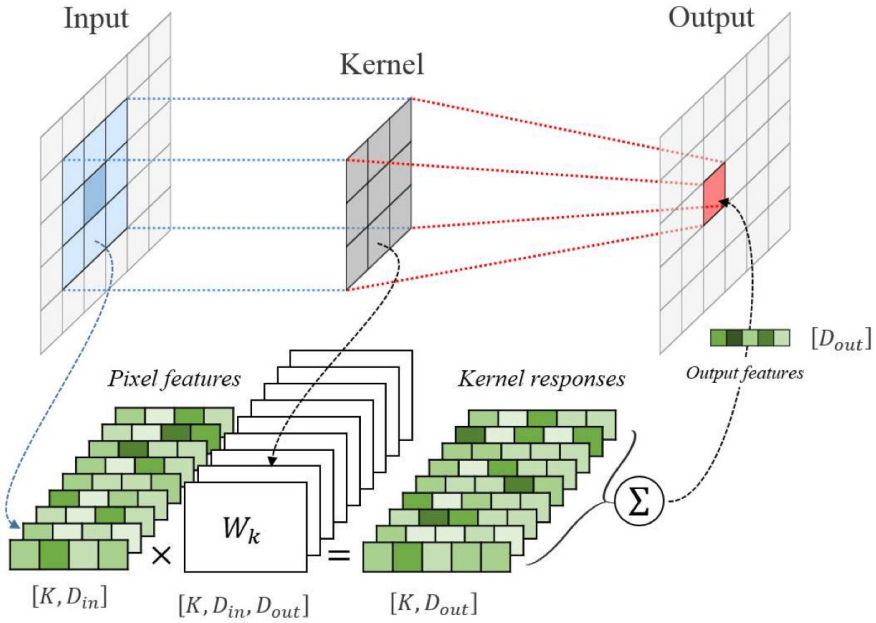
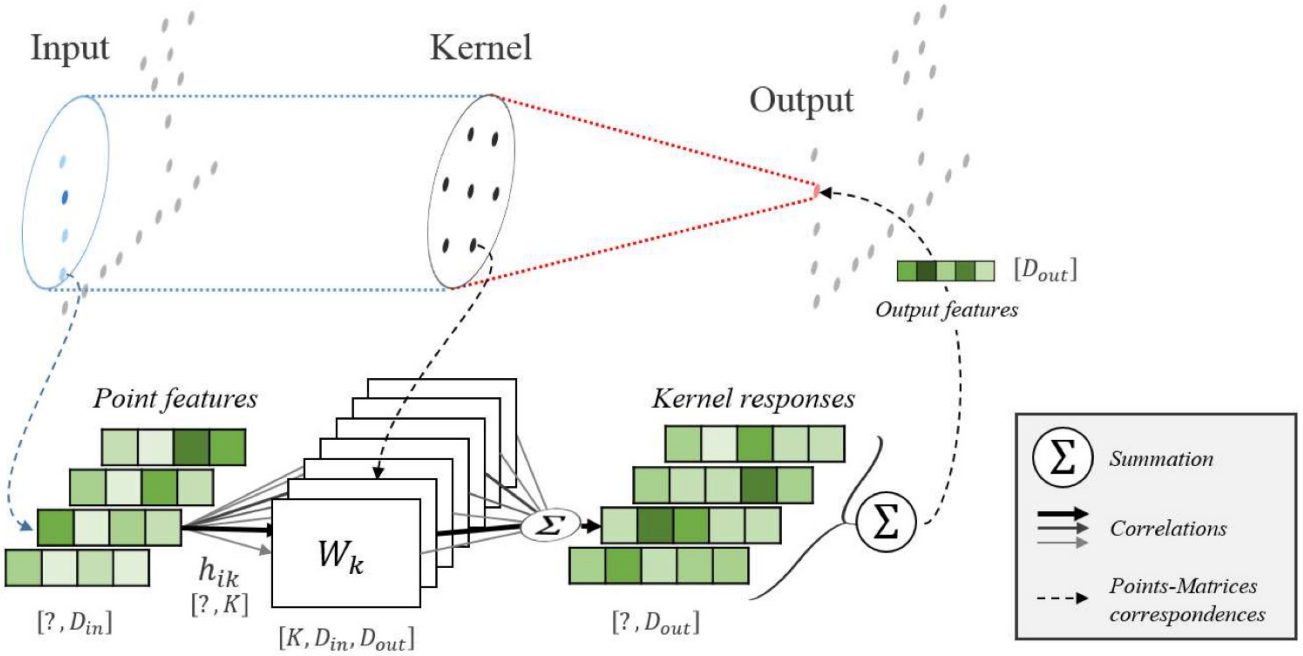


Image source: original publication, arXiv:1904.08889



(Standard) Image convolution

Pixel feature is assigned **one** weight (matrix) W_k by kernel alignment



Kernel point convolution

Input (point) feature is multiplied by **all** kernel weight matrices according to correlation weight h_{ik} depending on input and kernel point.

$$h\left(y_i, \widetilde{x}_k\right) = \max\left(0, 1 - \frac{\|y_i - \widetilde{x}_k\|}{\sigma}\right)$$

Image source: original publication, arXiv:1904.08889

KPConv networks

- Kernel point networks can be built, inspired by CNN, by replacing CNN by KPConv layers
- Point cloud subsampling is used, cell size is doubled with every pooling layer
- A variant, *deformable KPConv*, allows to learn kernel point positions

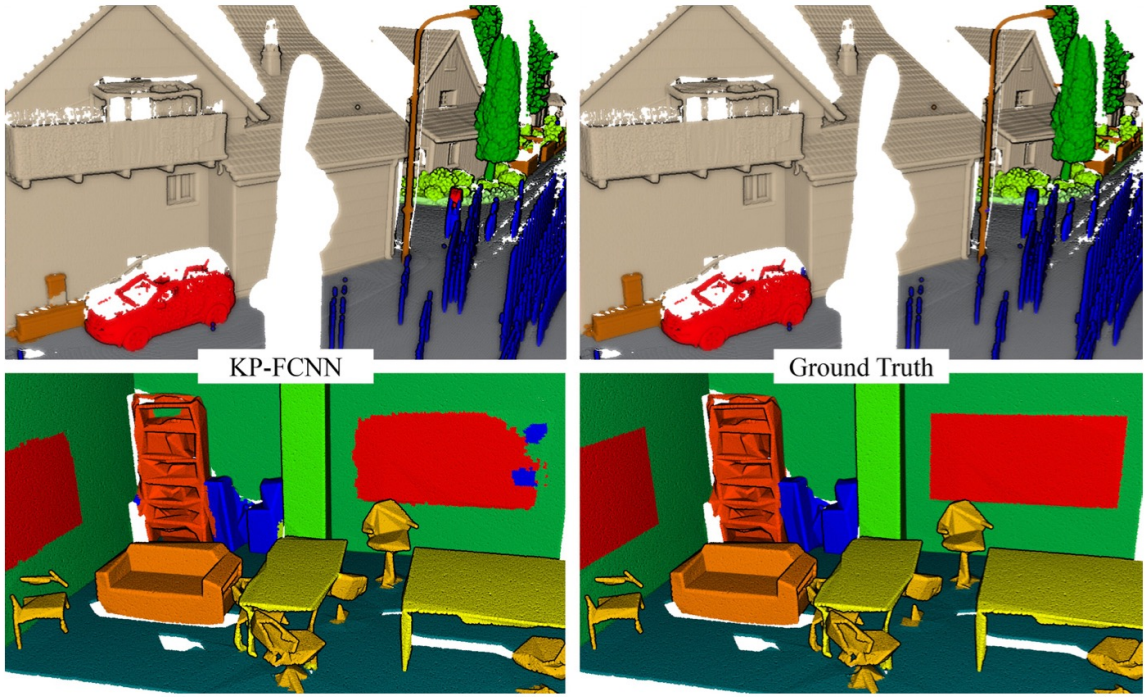


Figure 4. Outdoor and Indoor scenes, respectively from Semantic3D and S3DIS, classified by KP-FCNN with deformable kernels.

PointPillars
Uses pillars for detecting 3D objects and their bounding boxes

PointPillars

- CVPR 2019, see https://arxiv.org/abs/1812.05784v2
- Basic idea:
  - Operates on single rotation scans ('Velodyne' type)
  - Major idea: does not use voxels, but vertical columns (pillars)
  - Three stages:
    - Pillar Feature Net: Feature encoder network, point cloud → sparse pseudo-image
    - Backbone: 2D convolutional network, pseudo-image → high-level rep.
    - Detection head: 'single shot' bounding box detection and regression, high-level representation → 3D bounding boxes
- ("pillars" idea has been around in a similar way under the name "stixel representation")

Pillar feature net (1st stage)

- Ground plane is discretized (e.g. 0.16m × 0.16m grid)
- Each point (x, y, z) with reflectance r is first augmented by (x_c, y_c, z_c) (distance to centre of mass in the pillar) and (x_p, y_p) (offset from pillar centre) → dimension of data is $D = 9$
- Use at most P non-empty pillars per sample, N points per pillar
  - If there is more, randomly sample
  - If there is less, use zero padding
- Yields a dense tensor $D \times P \times N$
- A simplified version of PointNet is run, generating $C \times P$ output (C encoded features per pillar P)
- The features are projected back to the original location, resulting in a $C \times H \times W$ tensor, i.e. each location in the $H \times W$ 'canvas' stores C features.

PointPillars

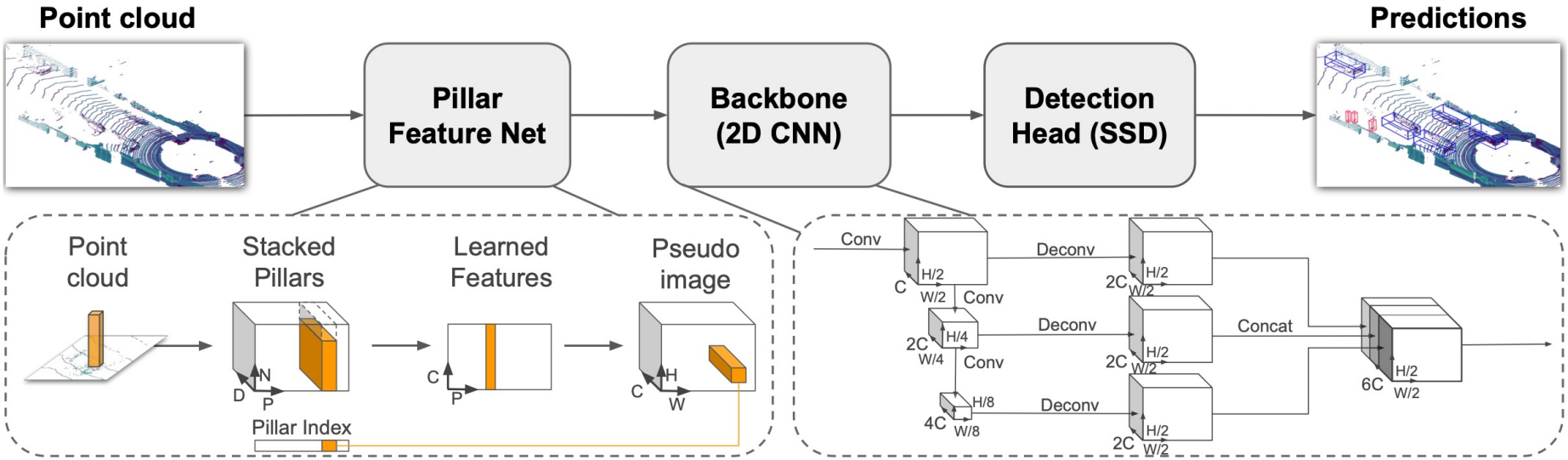


Image source: original publication, arXiv:1812.05784v2

Detection results

Figure 3. Qualitative analysis of KITTI results. We show a bird's-eye view of the lidar point cloud (top), as well as the 3D bounding boxes projected into the image for clearer visualization. Note that our method *only* uses lidar. We show predicted boxes for car (orange), cyclist (red) and pedestrian (blue). Ground truth boxes are shown in gray. The orientation of boxes is shown by a line connected the bottom center to the front of the box.

Image source: original publication, arXiv:1812.05784v2