


FACULTY OF MATHEMATICS,
PHYSICS AND INFORMATICS
Comenius University
Bratislava

3D Vision

Lecture 8: Pointcloud Processing

Ing. Viktor Kocur, PhD.

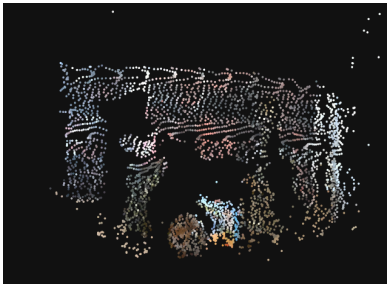
18.4.2023

Contents

Types of Pointcloud Based on Origin

We may encounter different types of point cloud data. We can have multiple different origins for such data.

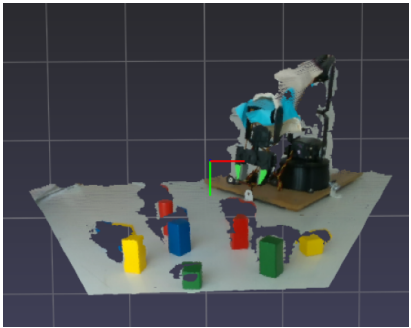
- Reconstruction from RGB images



Types of Pointcloud Based on Origin

We may encounter different types of point cloud data. We can have multiple different origins for such data.

- Reconstruction from RGB images
- Depth Cameras



Types of Pointcloud Based on Origin

We may encounter different types of point cloud data. We can have multiple different origins for such data.

- Reconstruction from RGB images
- Depth Cameras
- LiDAR/RADAR

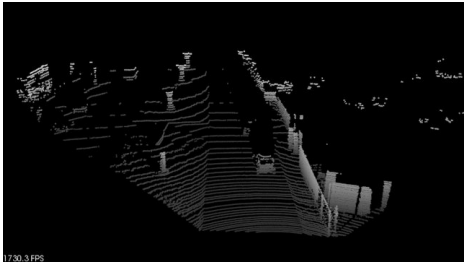


Image adopted from: Liang Xiao et al. “Hybrid conditional random field based camera-LIDAR fusion for road detection.” In: *Information Sciences* 432 (2018), pp. 543–558

Pointcloud

In general the pointcloud data can be stored as a set of triplets:

$$(X_i, Y_i, Z_i)^T, \tag{1}$$

or including the color or intensity information:

$$(X_i, Y_i, Z_i, R_i, G_i, B_i)^T. \tag{2}$$

Structured Pointclouds

In general we only have a set of the points and we do not know which points are neighbors for instance. However in many cases we may be able to know this information. Consider a pointcloud generated using stereo. We can then express each point as

$$(X_{i,j}, Y_{i,j}, Z_{i,j})^T \sim (x_{i,j}, y_{i,j}, Z_{i,j})^T, \tag{3}$$

where x, y are the coordinates in the image. We can obtain the the 3D coordinates if we known the intrinsic camera matrix K as

$$\begin{pmatrix} X_{i,j} \\ Y_{i,j} \\ Z_{i,j} \end{pmatrix} \sim ZK^{-1} \begin{pmatrix} x_{i,j} \\ y_{i,j} \\ 1 \end{pmatrix}. \tag{4}$$

Note that in many cases the x, y coordinates are not explicitly known instead only the depth values $Z_{i,j}$ are stored in an array.

Structured from Unstructured Pointcloud

In previous slide we assumed that the pointcloud came from a stereo pair. However we may create a structured pointcloud from an unstructured one by using a projection onto a surface.

In this step we may or may not use standard projective geometry onto a plane. We may instead opt for affine or orthographic projection or spherical projection.

Spherical Projection of LiDAR Pointcloud

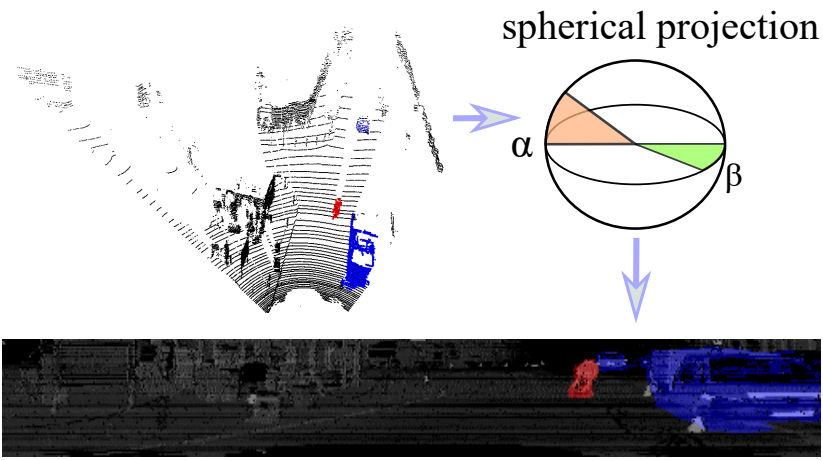


Image adopted from: Yuan Wang et al. “Pointseg: Real-time semantic segmentation based on 3d lidar point cloud.” In: *arXiv preprint arXiv:1807.06288* (2018)

Voxel Representation

We can also use a voxel representation. Voxels are organized in a 3D grid similar to pixels in 2D images. Each voxel may contain information about a presence or absence of a point from pointcloud or some other property such as intensity or color.

The issue with voxels is that they require lot of memory if we want good resolution. The space complexity of a voxel representation grows $O(n^3)$ with the resolution n. This is still a very limiting factor in terms of computation and storage.

Octrees

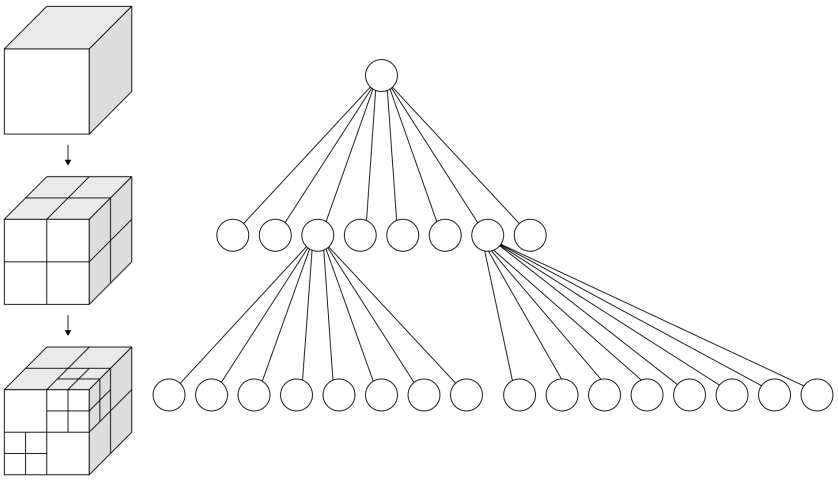


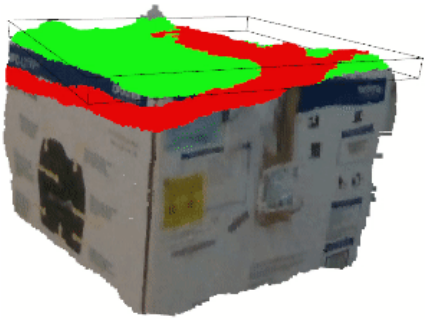
Image adopted from: WhiteTimberwolf. *Schematic drawing of an octree, a data structure of computer science*. https://commons.wikimedia.org/wiki/File:Octree2.svg. Online; accessed 12-April-2023. 2023

Basic Pointcloud Operations

We can perform some basic operations on the pointcloud. Since we can determine a neighborhood for a given point we can use this information for processing. Using the local neighborhood we can:

- Filter points
- Subsample the pointcloud
- Estimate surface normals
- Find keypoints and calculate their descriptors

RANSAC



Geometrical primitives as well as full models can be detected using RANSAC.

Image adopted from: Leo Mariga. *pyRANSAC-3D*. https://github.com/leomariga/pyRANSAC-3D. Online; accessed 12-April-2023. 2023

Aligning Pointclouds

Consider that we have two pointclouds C and C'. We want to find a transformation T such that it aligns the pointcloud C' to C. Given point correspondences we could find the transformation using a least-squares method. However, we often do not have good correspondences. We can therefore use the **iterative closest point** (ICP) algorithm:

1. Associate each point from C' with the closest point from C
2. Using these associations find the transformations T which in a leas squares manner
3. Transform C' using T
4. Repeat from 1

ICP usually requires a good initial estimate to work. There are many variants available. In structured environments we may calculate correspondences between points and planes/surfaces.

Deep Learning with Pointclouds

In this part of the lecture we will discuss some most common deep learning approaches when dealing with pointclouds. This section assumes familiarity with the basics of deep learning for computer vision. We will now cover the basic types of network structures used to deal with pointclouds and showcase more concrete examples in the following lectures.

With structured pointclouds we can treat them essentially as images on a plane and use approaches similar to the ones used on RGB images.

2D Convolutional Neural Networks

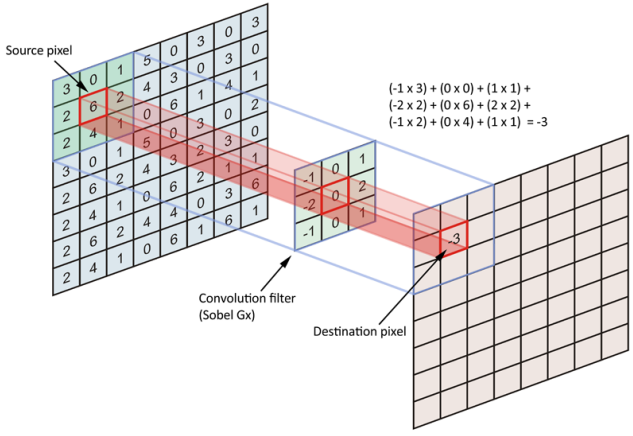


Image adopted from: Dynamic Stardust. *Convolution Operation*. https://datascience.stackexchange.com/questions/23183/why-convolutions-always-use-odd-numbers-as-filter-size. Online; accessed 13-April-2023. 2023

3D Convolutional Neural Networks

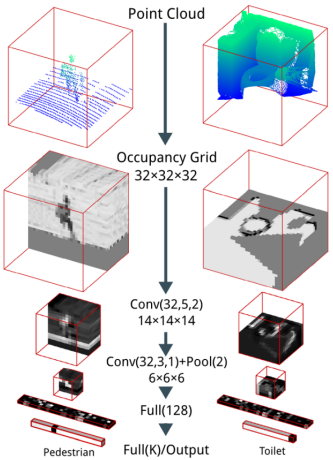


Image adopted from: Daniel Maturana and Sebastian Scherer. “Voxnet: A 3d convolutional neural network for real-time object recognition.” In: *2015 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE. 2015, pp. 922–928

Deep Learning with Unstructured Pointclouds

When it comes to unstructured pointclouds the points are essentially a set. In order for a network to work meaningfully we have two requirements:

- Permutation invariance - e.g. oder is not important
- Transformation equivariance - e.g. $f(T(x)) = T(f(x))$

Permutation Invariance

Since the input is a set of points it should not matter how we order the points when inputting them into a deep neural network. If we consider a function $f : \mathbb{R}^n \mapsto \mathbb{R}$ then we want the function to have the following property:

$$f(x_1, x_2, ...x_n) = f(x_{\pi_1}, x_{\pi_2}, ...x_{\pi_n}), \tag{5}$$

where $\pi_1, \pi_2, ...\pi_n$ is any permutation of $\{1, 2, ...n\}$. Such functions are called symmetric. Some examples:

$$f(\boldsymbol{x}) = \sum_{i}^{n} x_i \tag{6}$$

$$f(\boldsymbol{x}) = \max_i(\{x_i\}) \tag{7}$$

Constructing a Symmetric Function

Now consider functions $h : \mathbb{R}^m \mapsto \mathbb{R}^k$ and $g : \mathbb{R}^{k \cdot n} \mapsto \mathbb{R}^p$, $\gamma : \mathbb{R}^p \mapsto \mathbb{R}^q$ along with n input vectors $\boldsymbol{x}_i \in \mathbb{R}^m$. Then the following function

$$f(\boldsymbol{x}_1, \boldsymbol{x}_2, ...\boldsymbol{x}_n) = \gamma\left(g\left(h(\boldsymbol{x}_1), h(\boldsymbol{x}_2), ...h(\boldsymbol{x}_n)\right)\right), \tag{8}$$

is symmetric when g is symmetric.

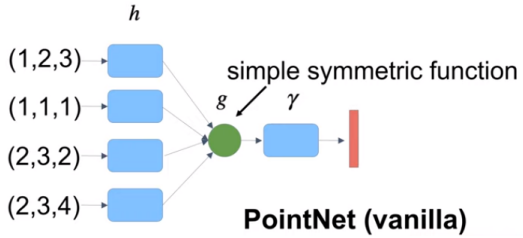


Image adopted from: Hao Su. *Deep Learning on Point Clouds*. https://www.youtube.com/watch?v=gm_oW0bdzHs. Online; accessed 13-April-2023. 2023

PointNet

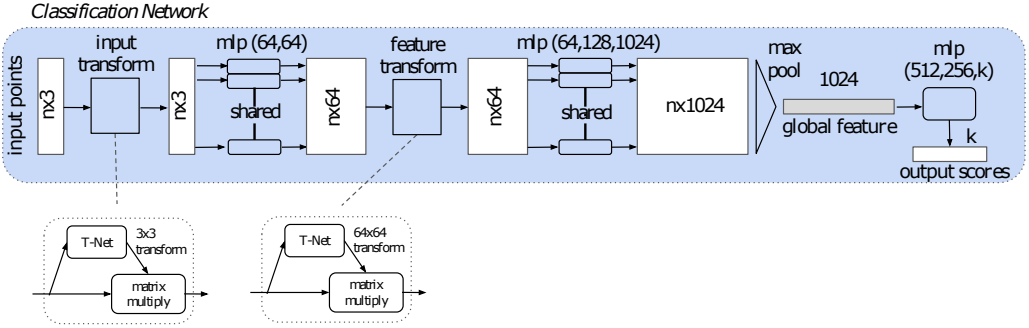


Image adopted from: Charles R Qi et al. “Pointnet: Deep learning on point sets for 3d classification and segmentation.” In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 652–660

PointNet - Semantic Segmentation

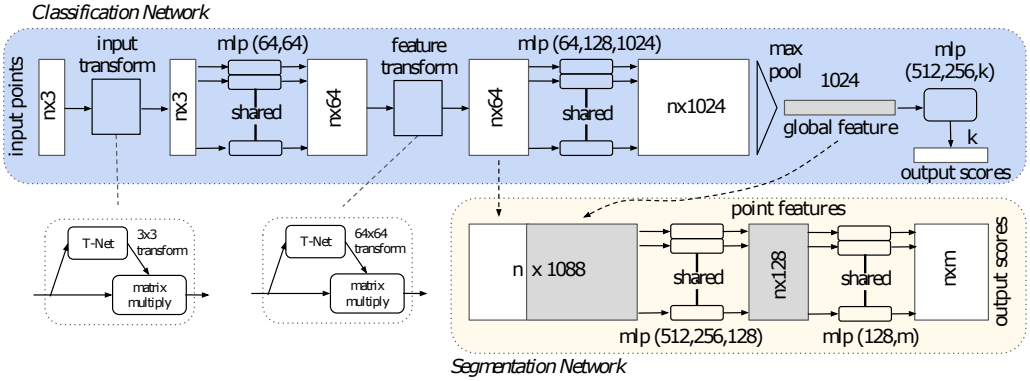


Image adopted from: Charles R Qi et al. “Pointnet: Deep learning on point sets for 3d classification and segmentation.” In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 652–660

PointNet - Not All Points Contribute Equally

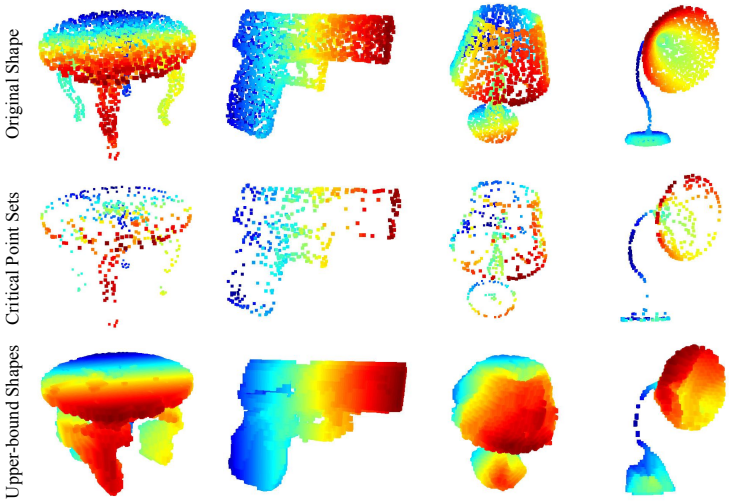
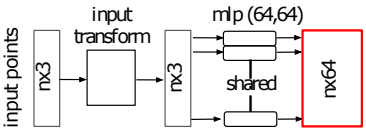


Image adopted from: Charles R Qi et al. “Pointnet: Deep learning on point sets for 3d classification and segmentation.” In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 652–660

PointNet - First Layer



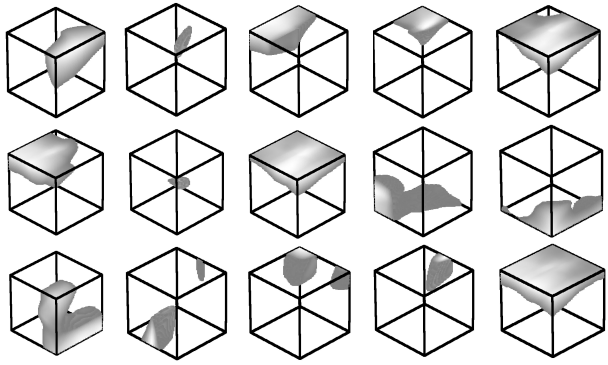


Image adopted from: Charles R Qi et al. “Pointnet: Deep learning on point sets for 3d classification and segmentation.” In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 652–660

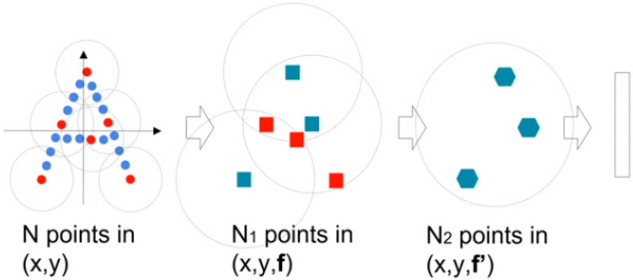
PointNet Limitations

There are two major limitations to PointNet:

- No local context for each point!
- Global features depend on the absolute coordinate. It is hard to generalize to unseen inputs!

Both of these are usually not an issue with convolutional CNNs.

PointNet++ - Multi-scale Pointnet



- Sample anchor points using Farthest Point Sampling
- Find neighborhoods of anchors
- Apply PointNet on each neighborhood (with normalization)

Image adopted from: Hao Su. *Deep Learning on Point Clouds*. https://www.youtube.com/watch?v=gm_oW0bdzHs. Online; accessed 13-April-2023. 2023

Pointnet++

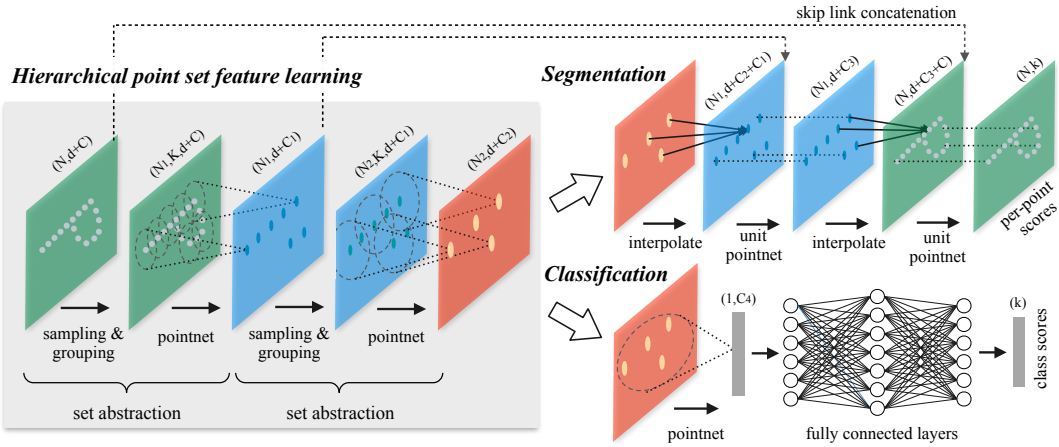


Image adopted from: Charles Ruizhongtai Qi et al. “Pointnet++: Deep hierarchical feature learning on point sets in a metric space.” In: *Advances in neural information processing systems* 30 (2017)

Graph Convolutional Networks

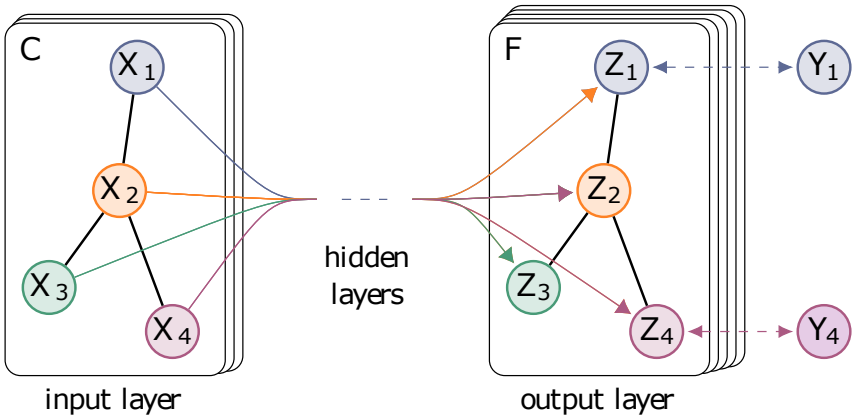


Image adopted from: Thomas N Kipf and Max Welling. “Semi-supervised classification with graph convolutional networks.” In: *arXiv preprint arXiv:1609.02907* (2016)

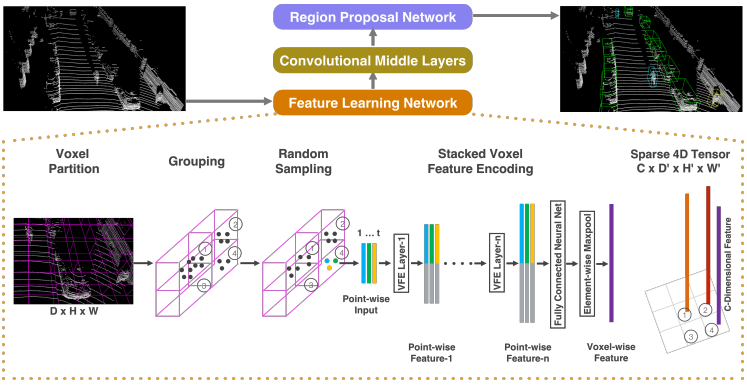
The formal expression of a GCN layer reads as follows:

$$H = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}X\Theta\right), \tag{9}$$

where H is the matrix of node representations $\boldsymbol{h}_u$, X is the matrix of node features $\boldsymbol{x}_u$, σ is an activation function (e.g., ReLU), $\tilde{A}$ is the graph **adjacency matrix** with the addition of self-loops, $\tilde{\mathbf{D}}$ is the graph degree matrix with the addition of self-loops, and Θ is a matrix of trainable parameters.

We can get $\tilde{A}$ from the standard adjacency matrix as $\tilde{A} = A + I$ and $\tilde{D} = \sum_{j \in V} \tilde{A}_{i,j}$. Also to obtain $M^{-\frac{1}{2}} = U\text{diag}(\frac{1}{\lambda_1^2}, \frac{1}{\lambda_2^2}, \ldots \frac{1}{\lambda_n^2})U^T$ if we consider the U to be composed of eigenvectors of M and λ_i are its eigenvalues.

VoxelNet



This network first uses PointNet to calculate features in each voxel and then continues with 3D CNN on sparse representation.

Image adopted from: Yin Zhou and Oncel Tuzel. “Voxelnet: End-to-end learning for point cloud based 3d object detection.” In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 4490–4499

Rotation Equivariance - Vector Neurons

We can deal with $\mathrm{SO}(3)$ invariance by having each neuron represent a 3-dimensional vector. Applications of linear layers are then equivariant to rotations.

Image adopted from: Congyue Deng et al. “Vector neurons: A general framework for so (3)-equivariant networks.” In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 12200–12209

Vector Neurons - Non-Linearity

Given an input vector-list feature $V \in \mathbb{R}^{C\times 3}$, for each output vector-neuron $\boldsymbol{v'} \in V'$ we learn two weight matrices $W \in \mathbb{R}^{1\times C}$ and $U \in \mathbb{R}^{1\times C}$, linearly mapping the input feature V to a feature $\boldsymbol{q} \in \mathbb{R}^{1\times 3}$ and a direction $\boldsymbol{k} \in \mathbb{R}^{1\times 3}$:

$$\boldsymbol{q} = WV, \quad \boldsymbol{k} = UV. \tag{10}$$

We then define the output vector neuron as:

$$\boldsymbol{v'} = \begin{cases} \boldsymbol{q} & \text{if } \langle \boldsymbol{q}, \boldsymbol{k} \rangle \geqslant 0 \\ \boldsymbol{q} - \left\langle \boldsymbol{q}, \frac{\boldsymbol{k}}{|\boldsymbol{k}|} \right\rangle \frac{\boldsymbol{k}}{|\boldsymbol{k}|} & \text{otherwise.} \end{cases} \tag{11}$$

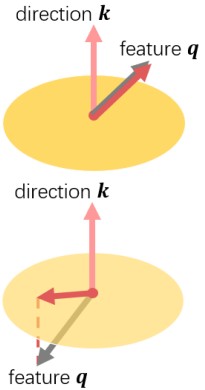
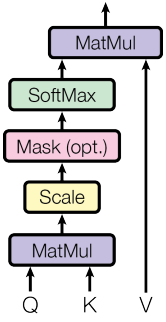
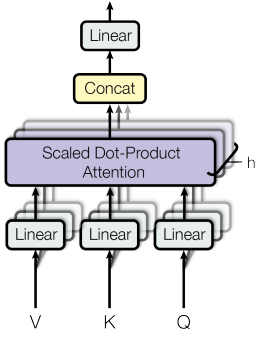
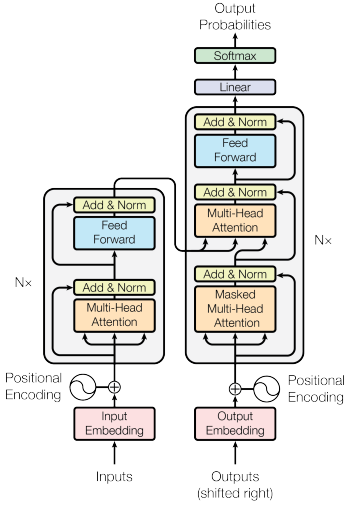


Image adopted from: Congyue Deng et al. “Vector neurons: A general framework for so (3)-equivariant networks.” In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 12200–12209

Transformers



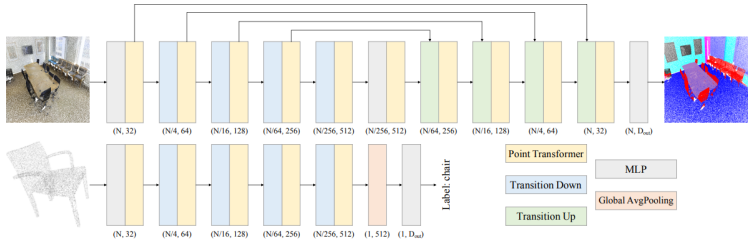


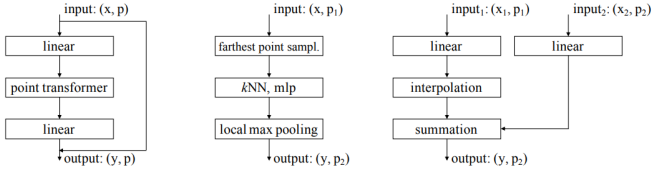


$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (12)$$

Image adopted from: Ashish Vaswani et al. “Attention is all you need.” In: *Advances in neural information processing systems* 30 (2017)

Point Transformer





(a) point transformer block (b) transition down (c) transition up

Image adopted from: Hengshuang Zhao et al. “Point transformer.” In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, pp. 16259–16268