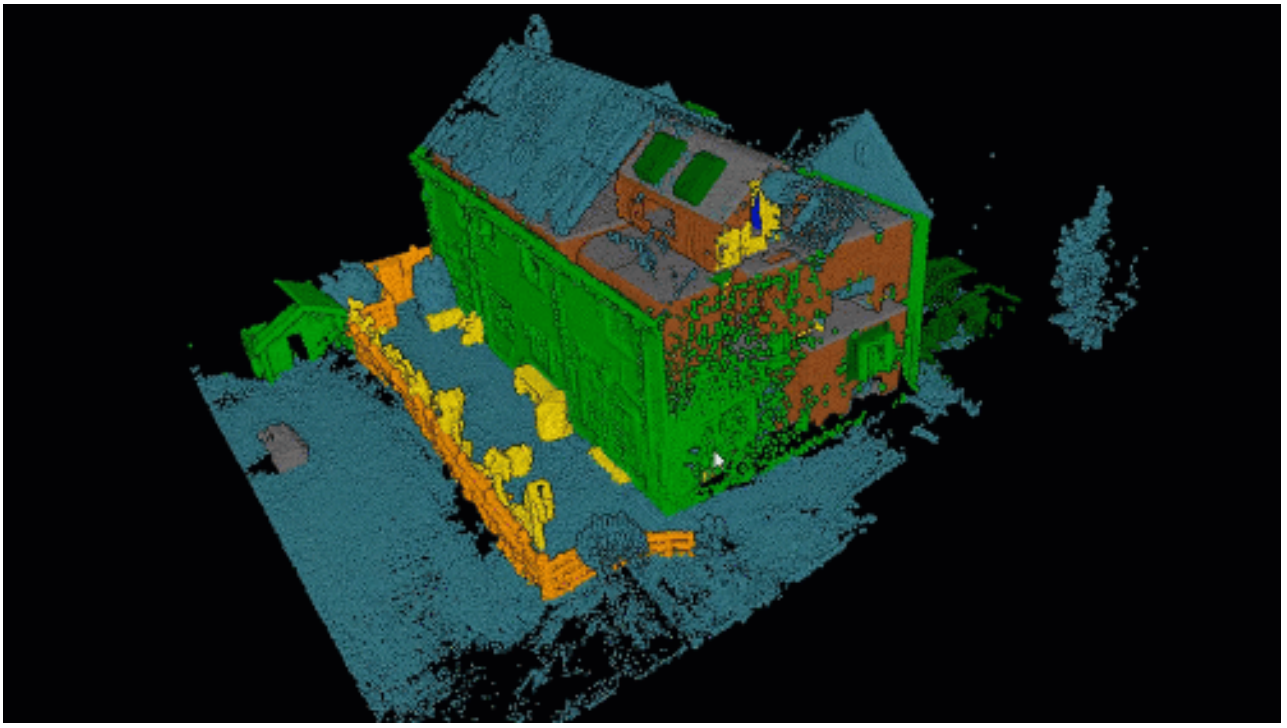
EE 046746 - Technion - Computer Vision

Tutorial 11 - Introduction to 3D Deep Learning



- Image source
- Cool video - Unreal Engine Point Clouds Feature: LiDAR point cloud support | Feature Highlight | Unreal Engine

Agenda

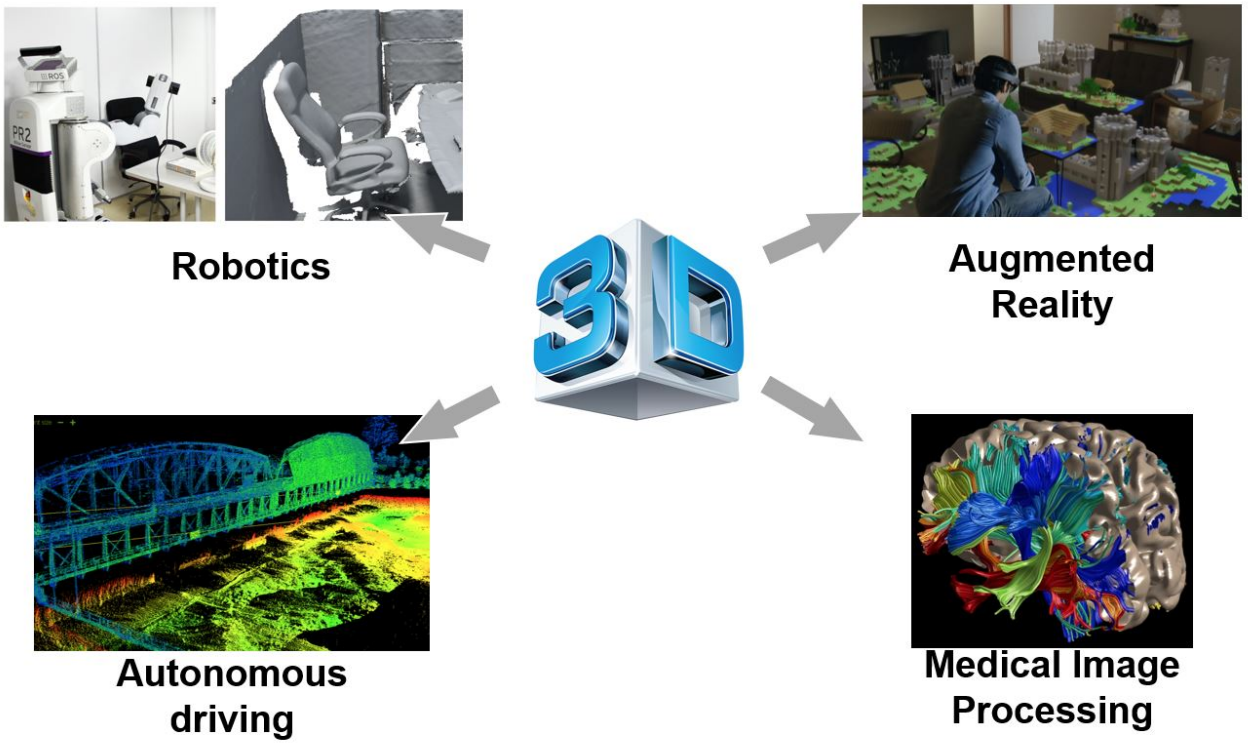
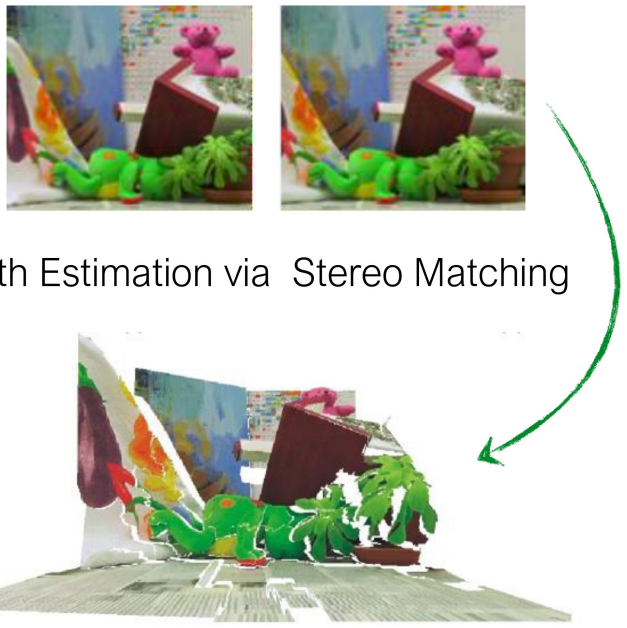


Image Source

Depth Cameras

- **Stereo Cameras - Last week**
- Time of flight

Stereo Cameras



Depth Cameras

- Stereo Cameras - Lecture with Anat
- **Time of flight**

Time of Flight Cameras

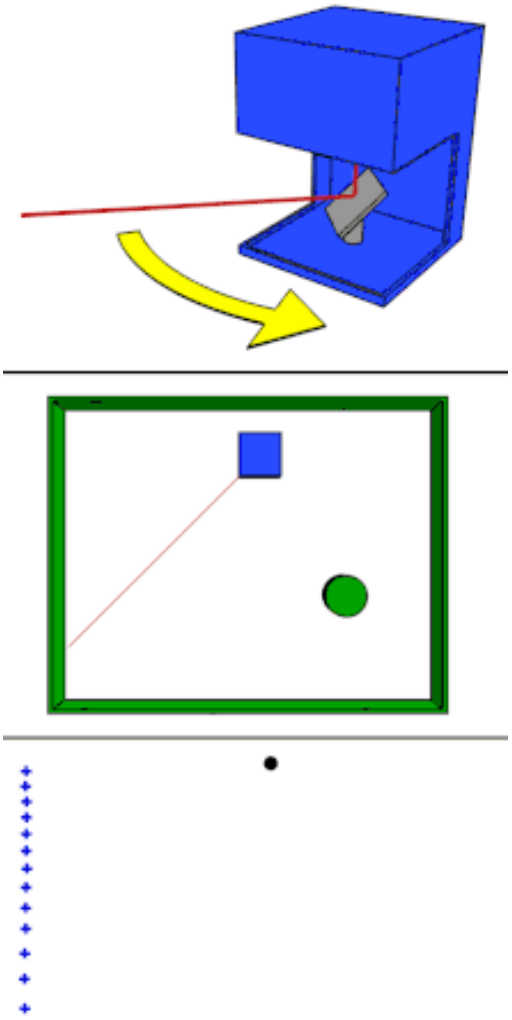
- Light travels at approximately a constant speed $c = 3 \times 10^8$ (meters per second).
- Measuring the time it takes for light to travel over a distance one can infer distance.
- Can be categorized into two types:
  1. Direct TOF - switch laser on and off rapidly.
  2. Indirect TOF - send out modulated light, then measure phase difference to infer depth.

Time of Flight Cameras

1. Direct - TOF

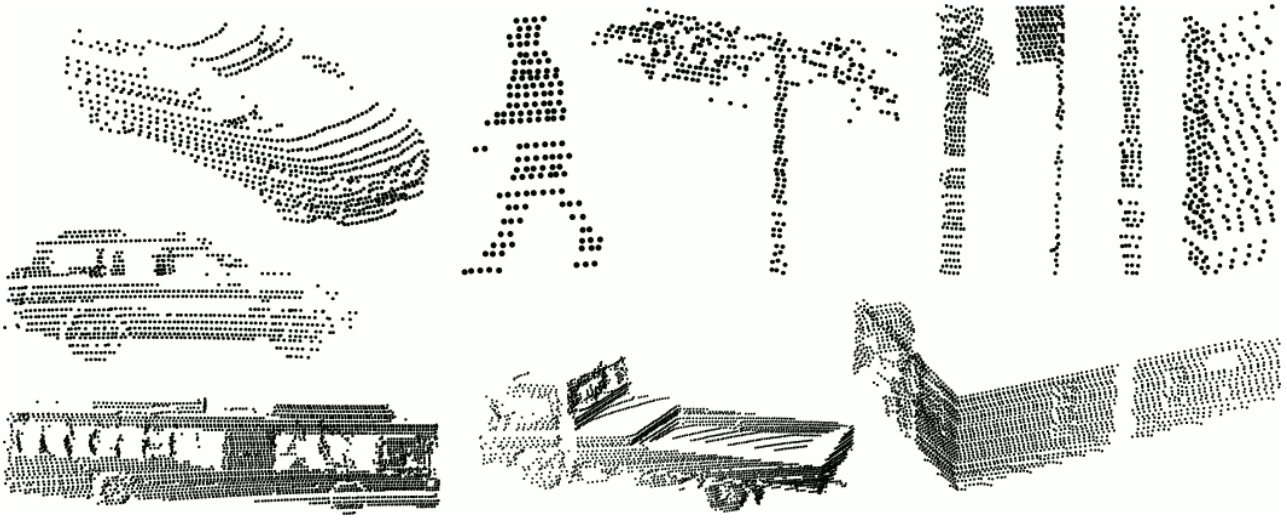
- **Li**ght **D**etection **A**nd **R**anging (LiDAR) probably best example in computer vision and robotics.
- High-energy light pulses limit influence of background illumination.
- However, difficulty to generate short light pulses with fast rise and fall times.
- High-accuracy time measurement required.
- Prone to motion blur.
- Sparser as objects grow in distance.

Time of Flight Cameras



Gif source - Wikipedia

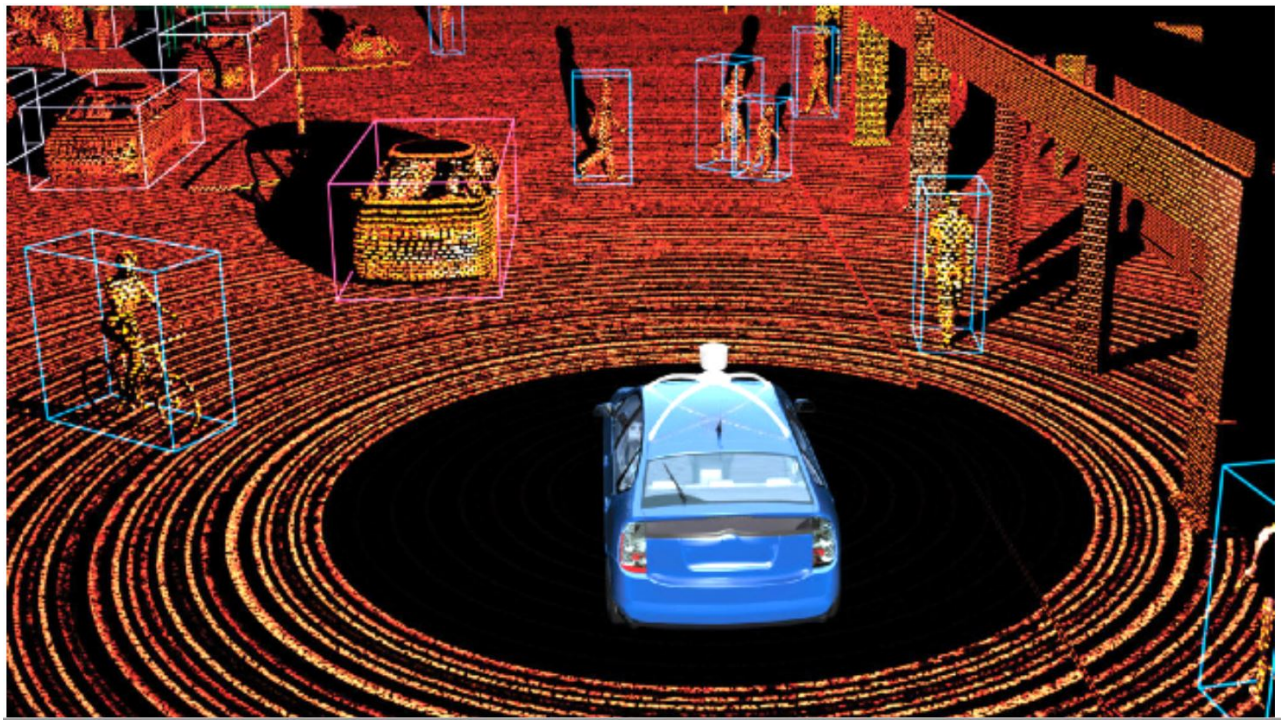
Time of Flight Cameras



Sydney Dataset

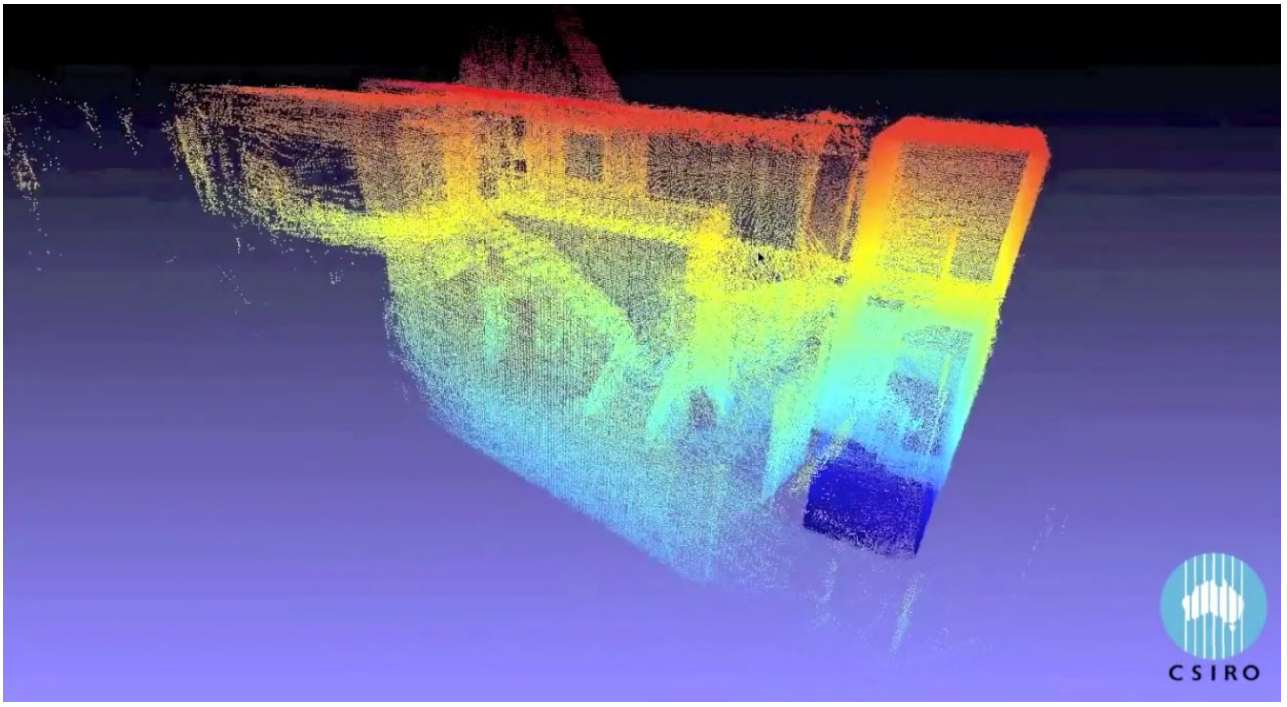
Time of Flight Cameras

Autonomous Car - LiDAR



Time of Flight Cameras

SLAM + LIDAT - Zebedee

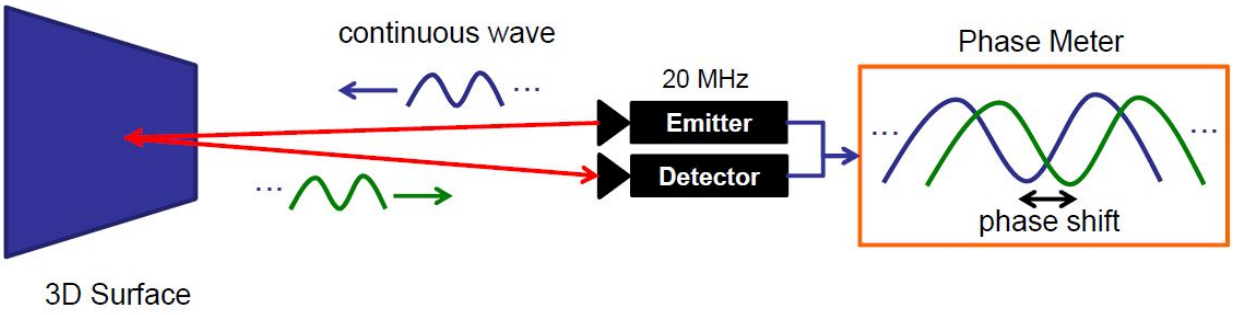


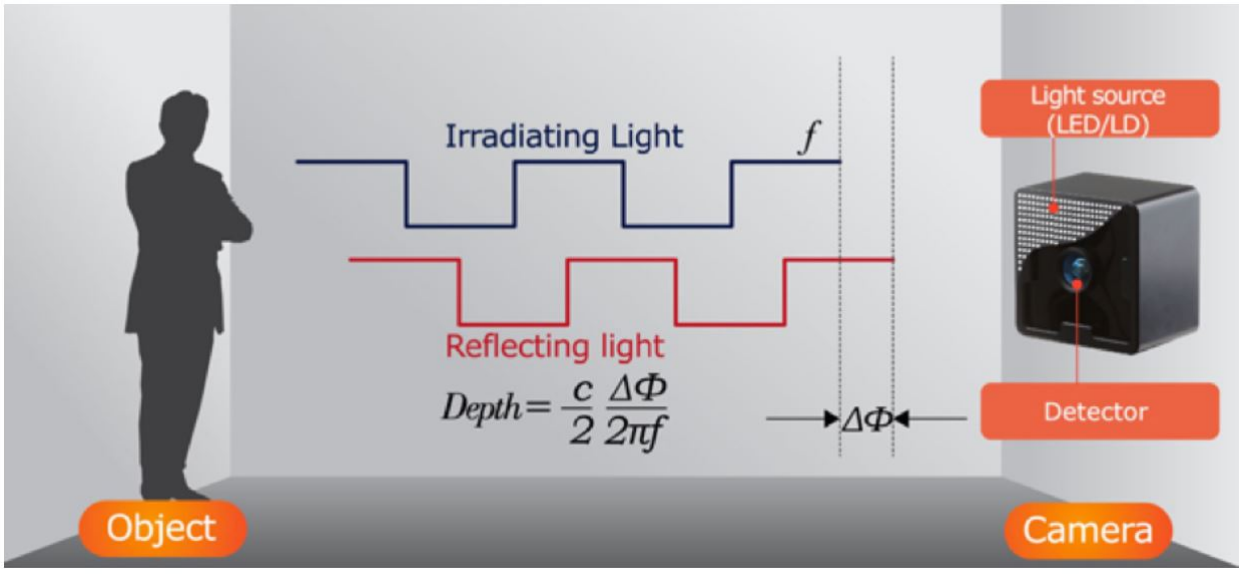
zebedee

Time of Flight Cameras

2. Indirect - TOF

- Continuous light waves instead of short light pulses.
- Modulation in terms of frequency of sinusoidal waves.
- Detected wave after reflection has shifted phase.
- Phase shift proportional to distance from reflecting surface.

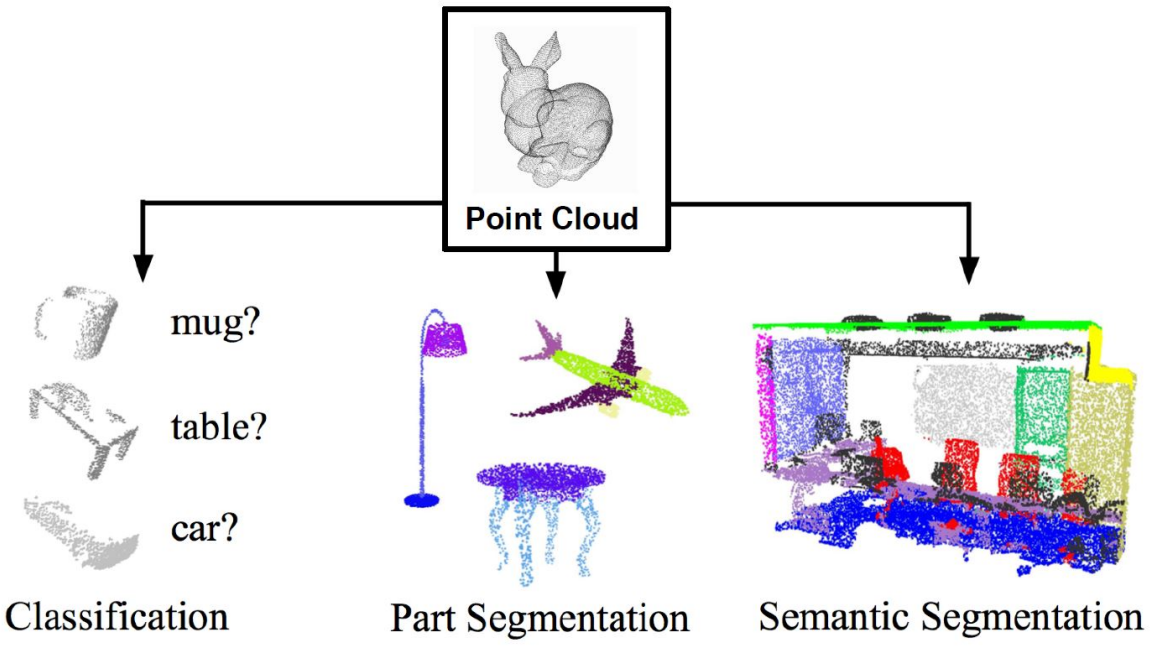




3D Deep Learning

- Representation
- Architectures

3D Deep Learning: Point Clouds?



- Image Source

3D Deep Learning: Point Clouds?

- Calssification
- Semantic segmentation
- Part segmentation
  - Each point belongs to a specific part of the object
- ...

3D Deep Learning: Point Clouds?

Questions

- What are the differences between 2D image and a point cloud?
- Why it might be hard to feed a point cloud as neural network input?
- What are the benefits of using a point cloud?

3D Deep Learning: Point Clouds?

NA

Point Clouds Problems

- Point Clouds Vary in Size (not constant)
- Unordered Input
  - Data is unstructured (no grid)
  - Data is invariant to point ordering (permutations)

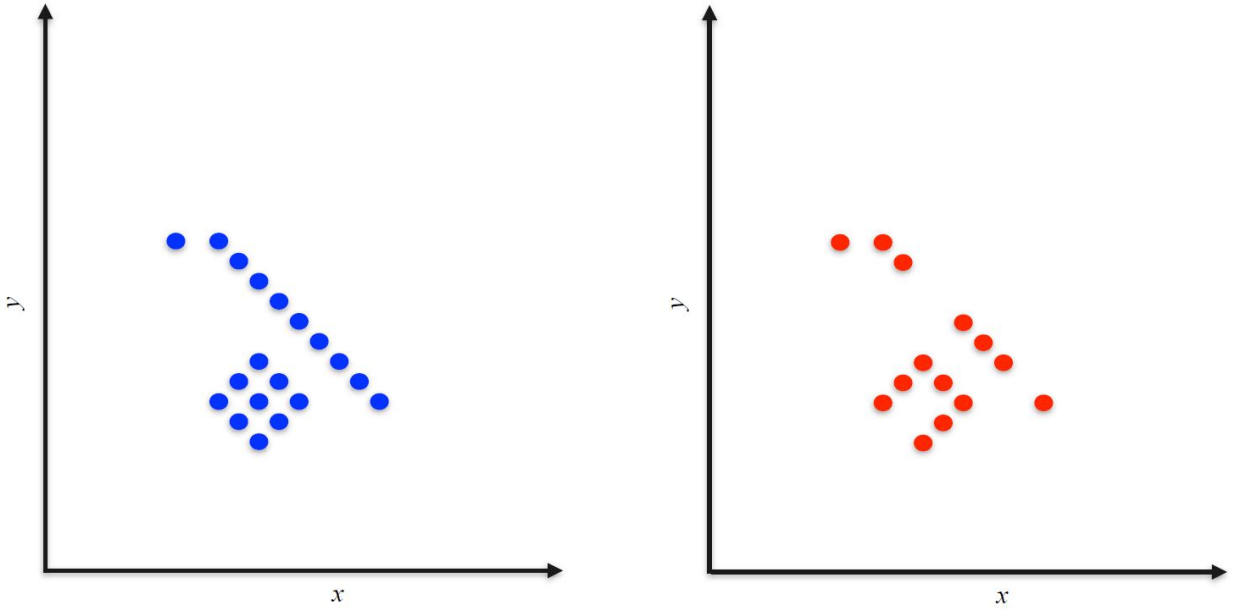
3D Deep Learning: Point Clouds?

Point Clouds Problems

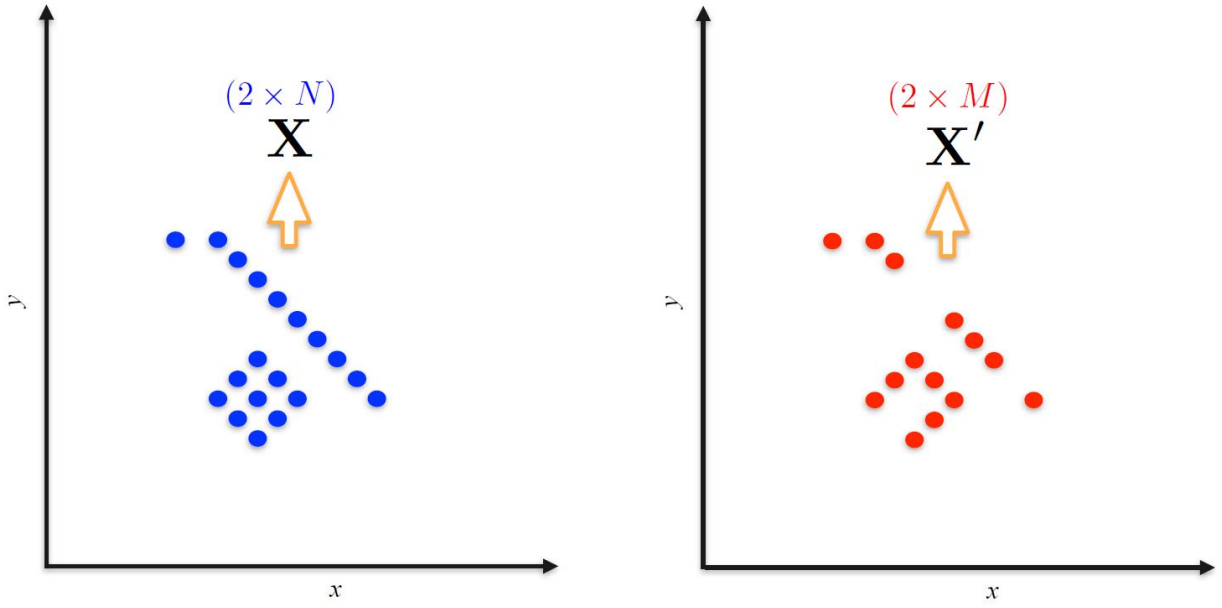
- Missing data
- Noise
- Rotations

3D Deep Learning: Point Clouds?

Problem - Point Clouds Vary in Size



3D Deep Learning: Point Clouds?

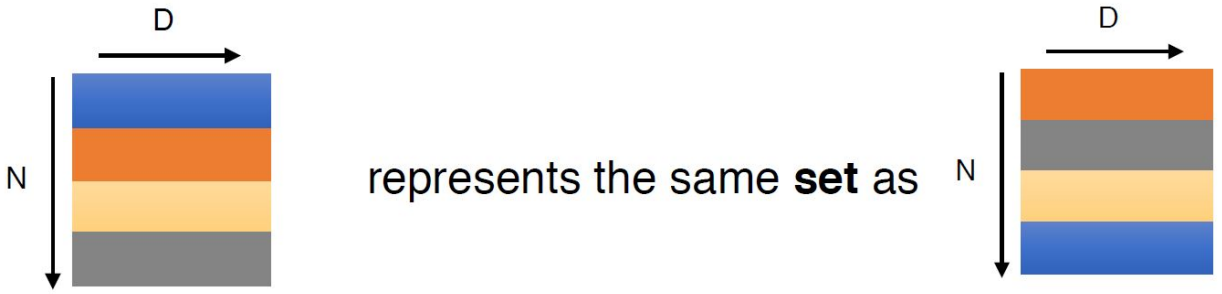


- Different point clouds represent the same object

3D Deep Learning: Point Clouds?

Problem - Unordered Input

Point cloud: N **orderless** points, each represented by a D dim vector



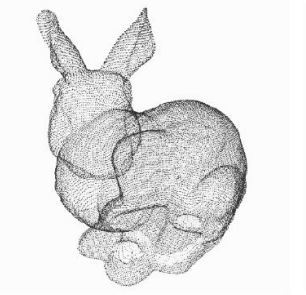
- How many semi-equal representations?
  - **Model needs to be invariant to $N!$ permutations**

3D Deep Learning: Point Clouds?

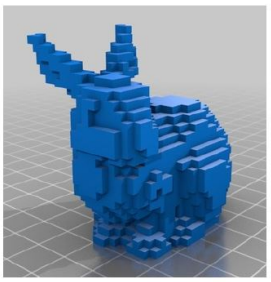
Alternative 3D Representations

Solution:

- Convert the raw point clouds into Voxels or multiple 2D RGB(D) images



Point Cloud



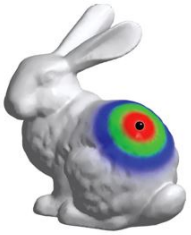
Volumetric

Projected View
RGB(D)

3D Deep Learning: Point Clouds?

- Other 3D representation (not in this course):

Part Assembly



Mesh (Graph CNN)

$$F(x) = 0$$

Implicit Shape

Voxelization

Idea: generalize 2D convolutions to regular 3D grids

- The straightforward approach: transform the point clouds into a voxel grid by rasterizing and use 3D CNNs

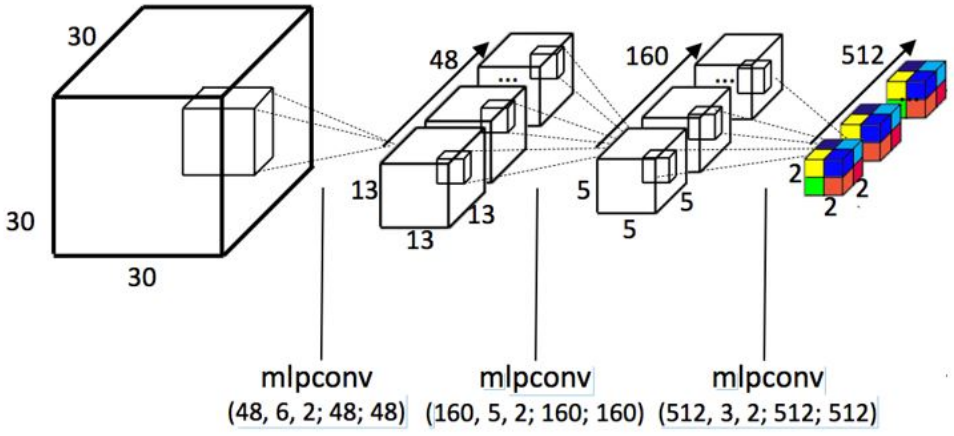
Voxel grid is a 3D grid of equal size volumes (voxels), can be occupied by:

- Binary 0/1 - Is there any point within the voxel?
- Weighted - The amount of point located within each voxel

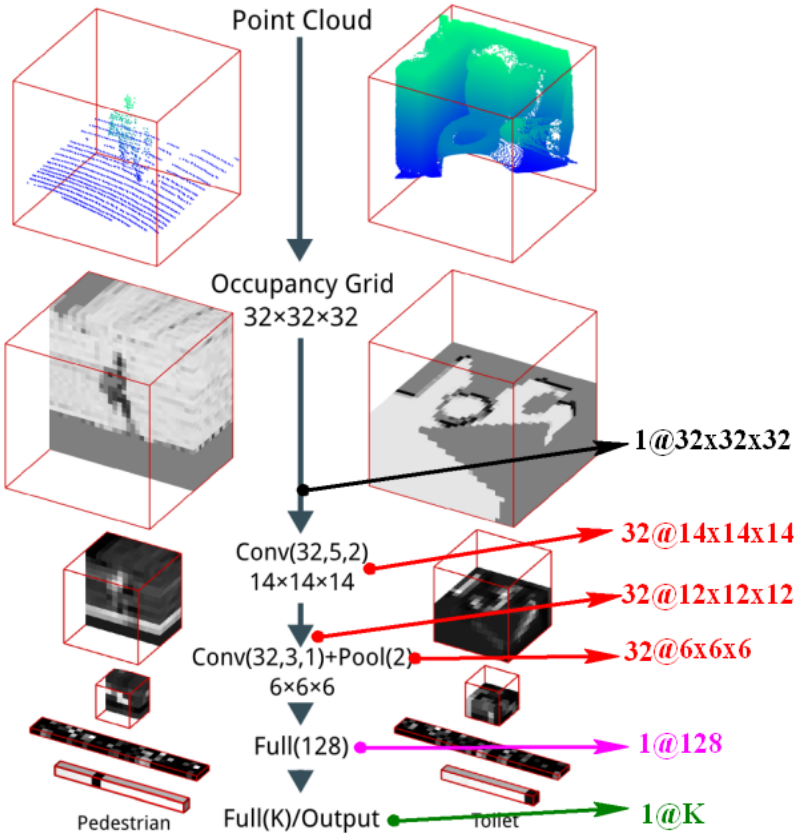
Usually we use binary occupancy

Voxelization

3D convolution uses 4D kernels



Voxelization



- Image Source

Voxelization

Voxelization Problems

- Large memory cost
- Slow processing time
- Limited spatial resolution
- Quantization artifacts

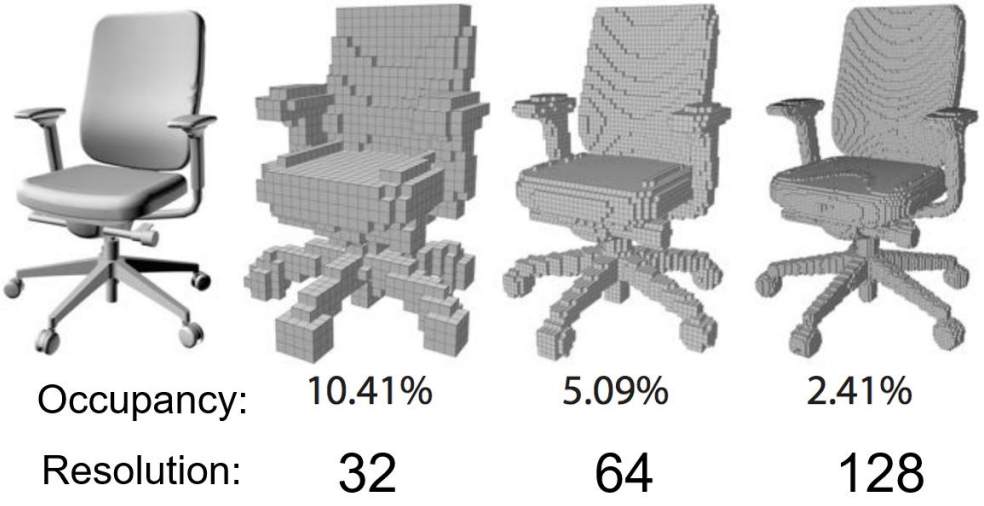
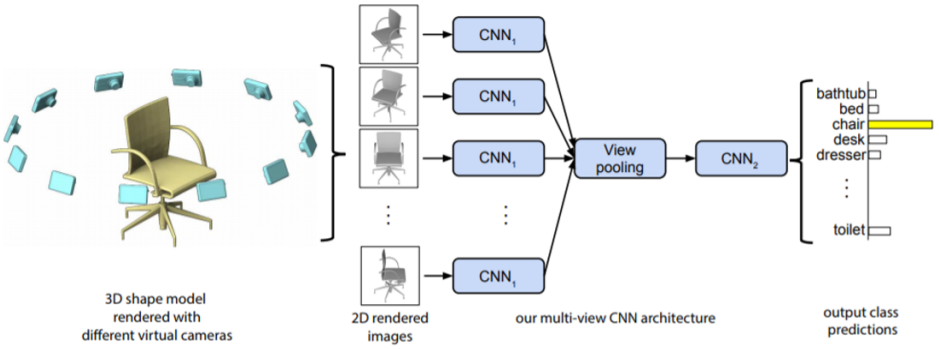


Image Source

Multi-View Approach

Idea: Transfrom the problem into a well known domain (3D→2D)

- The multi-view approach: project multiple views to 2D and use CNN to process
  - How many views do we need? (Another hyper parameter)



- CNN_1 - We can use pre-trained networks to extract features followed by fine tuned layers
- Image Source

Apply Deep Learning Directly on 3D Point Clouds

Idea: Most of the raw 3D data are point clouds - Solve the problems!

Note: Point Clouds Problems:

- Point Clouds Vary in Size (not constant)
- Unordered Input
  - Data is unstructured (no grid)
  - Data is invariant to point ordering (permutations)

PointNet

Permutation Invariance: Symmetric Function

$$f(x_1, x_2, \ldots, x_n) \equiv f(x_{\pi_1}, x_{\pi_2}, \ldots, x_{\pi_n}),\ x_i \in R^D$$

π is a different permutation

PointNet

Example:

$$f(x_1, x_2, \ldots, x_n) = max\{x_1, x_2, \ldots, x_n\}$$

$$f(x_1, x_2, \ldots, x_n) = x_1 + x_2 + \cdots + x_n$$

PointNet

- How can we construct a family of symmetric functions by neural networks?

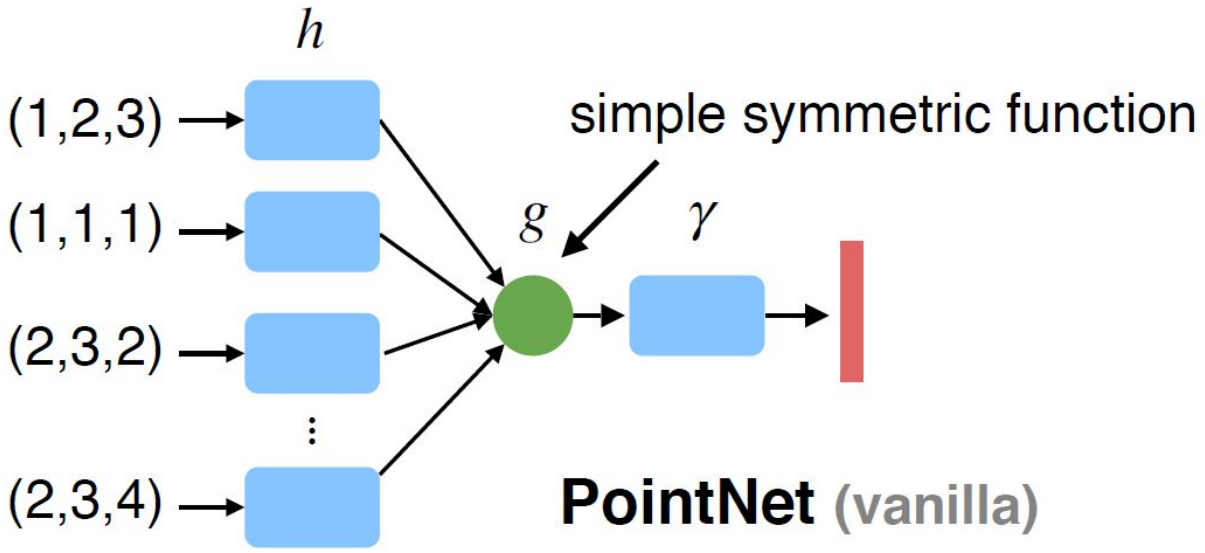
PointNet

Observe:

$$f(x_1, x_2, \ldots, x_n) = \gamma \circ g(h(x_1), \ldots, h(x_n))$$

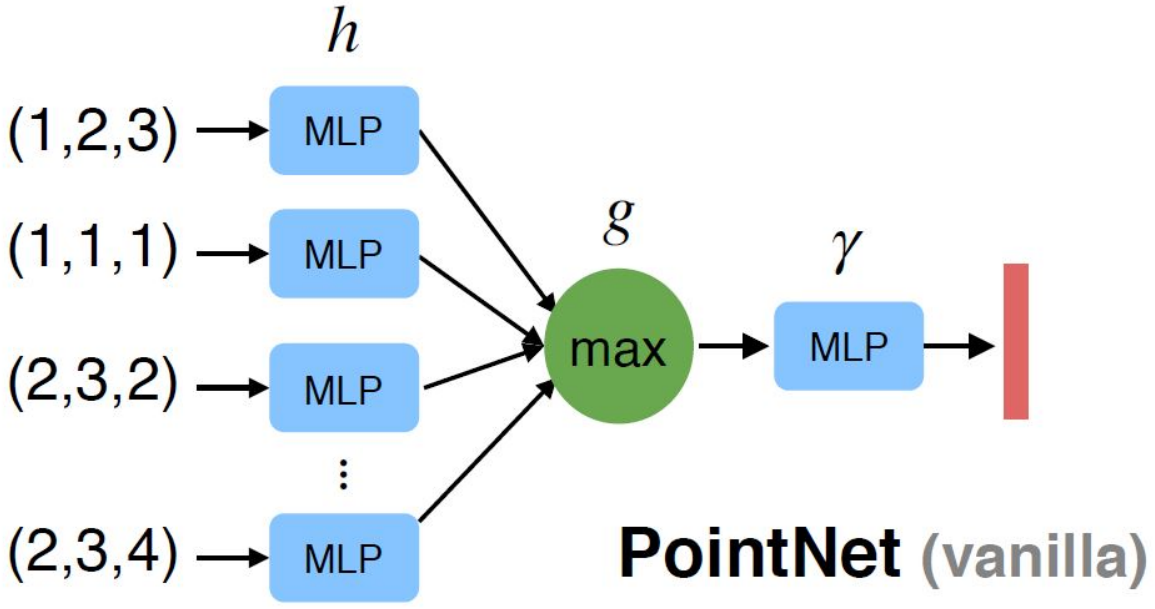
is symmetric if g is symmetric

PointNet



Basic PointNet Architecture

Empirically, we use multi-layer perceptron (MLP) and max pooling:



Basic PointNet Architecture

Input MLP:

$$h(x_i) : R^3 \rightarrow R^D$$

We can look at it as D functions $\{h_k\}_{k=1}^{D}$ operate on each point where

$$h_k(x_i) : R^3 \rightarrow R^1$$

Pooling layer:

$$g(h(x_1), \ldots, h(x_n)) : R^{N \times D} \rightarrow R^D$$

We apply the pooling over all points for each function h_k.

$$g(h_k(x_1), \ldots, h_k(x_n)) : R^{N \times 1} \rightarrow R^1$$

Classification MLP:

$$\gamma \circ g(h(x_1), \ldots, h(x_n)) : R^D \rightarrow R^{\text{Num Classes}}$$

Basic PointNet Architecture

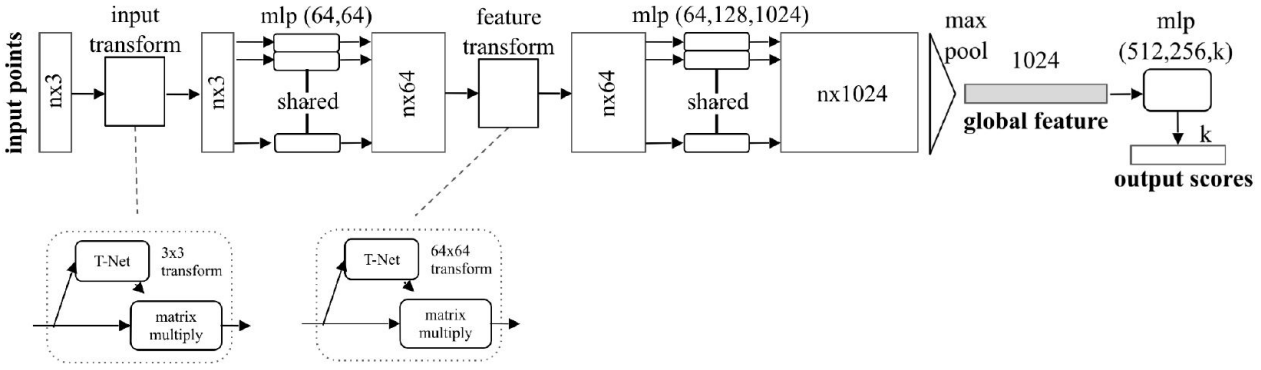
Shared mlp implementation "trick":

- Use conv layers : Number of filters C_{out}, each filter size is $1 \times C_{in}$.
- Input: $R^{N \times C_{in}}$
- Output: $R^{N \times C_{out}}$
- Main idea is to cast to 1D convolutions:
  - shift invariance in the N dimension takes care of weight sharing.

MLP:

$$h : R^{C_{in}} \rightarrow R^{C_1} \rightarrow \cdots \rightarrow R^{C_{out}}$$

PointNet Classification Network



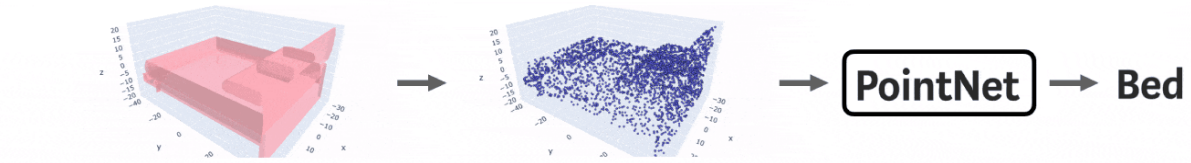
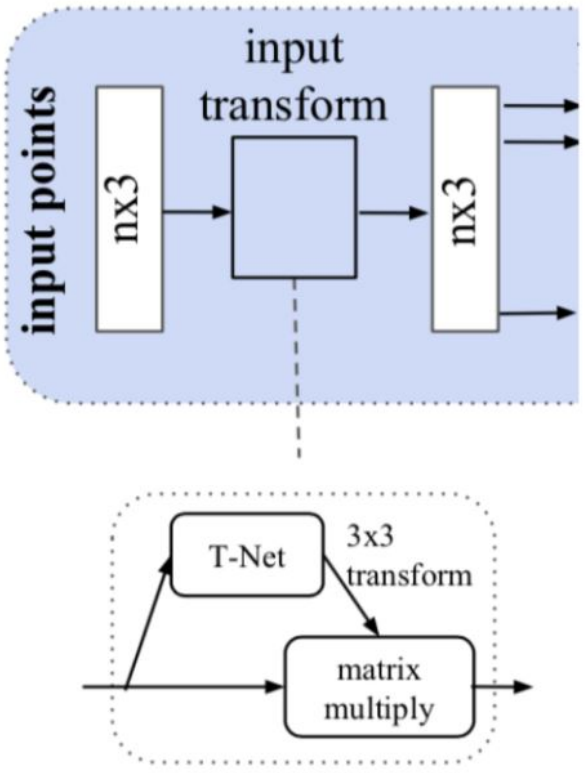


Image source

Transformation Invariance



- Learn transformation matrix to improve task performance:
  - We want our network to be invariant to rigid transformations of the object.
  - Practically, more augmentation over the training dataset also solved the transformation invariance.

Image source

Code Example

- Code based on example by Nikita Karaev

In [7]:
```
# imports for the tutorial
import numpy as np
import matplotlib.pyplot as plt
import time

# pytorch imports
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import DataLoader, Dataset, ConcatDataset
```

In [7]:
```
class Tnet(nn.Module):
   def __init__(self, k=3):
      super().__init__()
      self.k=k
      self.conv1 = nn.Conv1d(k,64,1)
      self.conv2 = nn.Conv1d(64,128,1)
      self.conv3 = nn.Conv1d(128,1024,1)
      self.fc1 = nn.Linear(1024,512)
      self.fc2 = nn.Linear(512,256)
      self.fc3 = nn.Linear(256,k*k)

      self.bn1 = nn.BatchNorm1d(64)
      self.bn2 = nn.BatchNorm1d(128)
```

```
        self.bn3 = nn.BatchNorm1d(1024)
        self.bn4 = nn.BatchNorm1d(512)
        self.bn5 = nn.BatchNorm1d(256)


    def forward(self, input):
        # input.shape == (bs,n,3)
        bs = input.size(0)
        xb = F.relu(self.bn1(self.conv1(input)))
        xb = F.relu(self.bn2(self.conv2(xb)))
        xb = F.relu(self.bn3(self.conv3(xb)))
        pool = nn.MaxPool1d(xb.size(-1))(xb)
        flat = nn.Flatten(1)(pool)
        xb = F.relu(self.bn4(self.fc1(flat)))
        xb = F.relu(self.bn5(self.fc2(xb)))

        #initialize as identity
        init = torch.eye(self.k, requires_grad=True).repeat(bs,1,1)
        if xb.is_cuda:
            init=init.cuda()
        matrix = self.fc3(xb).view(-1,self.k,self.k) + init
        return matrix
```

In [7]:

```
class Transform(nn.Module):
    def __init__(self):
        super().__init__()
        self.input_transform = Tnet(k=3)
        self.feature_transform = Tnet(k=64)
        self.conv1 = nn.Conv1d(3,64,1)
        self.conv2 = nn.Conv1d(64,128,1)
        self.conv3 = nn.Conv1d(128,1024,1)
        self.bn1 = nn.BatchNorm1d(64)
        self.bn2 = nn.BatchNorm1d(128)
        self.bn3 = nn.BatchNorm1d(1024)

    def forward(self, input):

        matrix3x3 = self.input_transform(input)
        xb = torch.bmm(torch.transpose(input,1,2), matrix3x3).transpose(1,2) # batch matrix multiplication
        xb = F.relu(self.bn1(self.conv1(xb)))
        matrix64x64 = self.feature_transform(xb)
        xb = torch.bmm(torch.transpose(xb,1,2), matrix64x64).transpose(1,2)

        xb = F.relu(self.bn2(self.conv2(xb)))
        xb = self.bn3(self.conv3(xb))
        xb = nn.MaxPool1d(xb.size(-1))(xb)
        output = nn.Flatten(1)(xb)
        return output, matrix3x3, matrix64x64
```

In [7]:

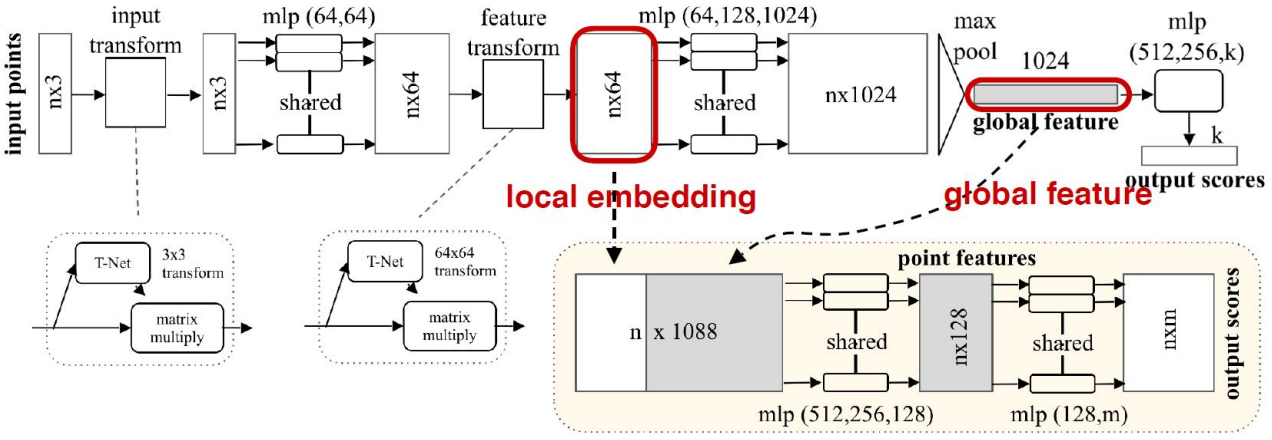
```
class PointNet(nn.Module):
    def __init__(self, classes = 10):
        super().__init__()
        self.transform = Transform()
        self.fc1 = nn.Linear(1024, 512)
        self.fc2 = nn.Linear(512, 256)
        self.fc3 = nn.Linear(256, classes)


        self.bn1 = nn.BatchNorm1d(512)
        self.bn2 = nn.BatchNorm1d(256)
        self.dropout = nn.Dropout(p=0.3)
        self.logsoftmax = nn.LogSoftmax(dim=1)

    def forward(self, input):
        xb, matrix3x3, matrix64x64 = self.transform(input)
        xb = F.relu(self.bn1(self.fc1(xb)))
        xb = F.relu(self.bn2(self.dropout(self.fc2(xb))))
        output = self.fc3(xb)
        return self.logsoftmax(output), matrix3x3, matrix64x64
```

Results - Classification

	input	#views	accuracy avg. class	accuracy overall
SPH [12]	mesh	-	68.2	
3DShapeNets [29]	volume	1	77.3	84.7
VoxNet [18]	volume	12	83.0	85.9
Subvolume [19]	volume	20	86.0	**89.2**
LFD [29]	image	10	75.5	-
MVCNN [24]	image	80	**90.1**	-
Ours baseline	point	-	72.6	77.4
Ours PointNet	point	1	86.2	**89.2**

3D CNNs

PointNet Segmentation Network



- Extract local features - Describes each point separately
- Extract global feature - Describes the entire point cloud
- Concatenate the local and global features and feed them into a shared mlp:
  - The mlp learns to process the point feature according to a condition.
  - The condition is described by the global feature vector.

Semantic Scene Parsing

Image Source

- Colab notebooks for:
  - PointNet Classification
  - PointNet Segmentation

Results - Robustness to Missing Data (Classification example)

- Why is PointNet so robust to missing data?

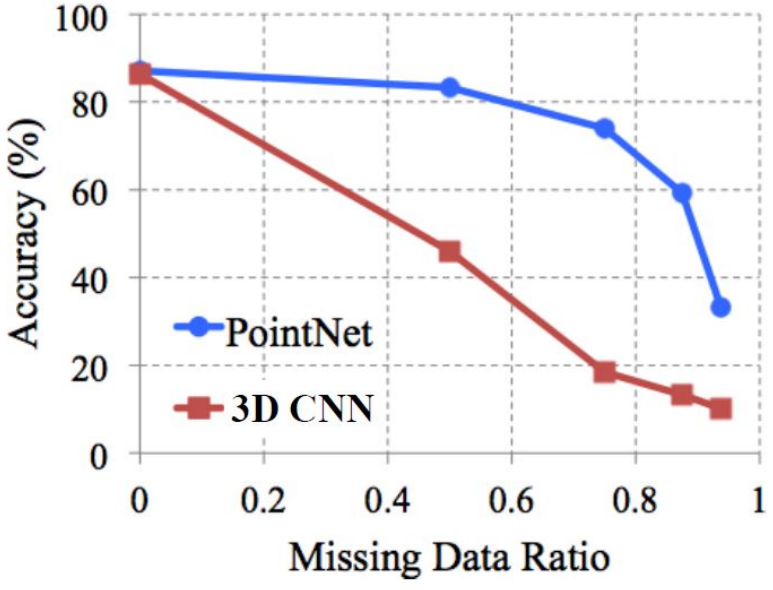
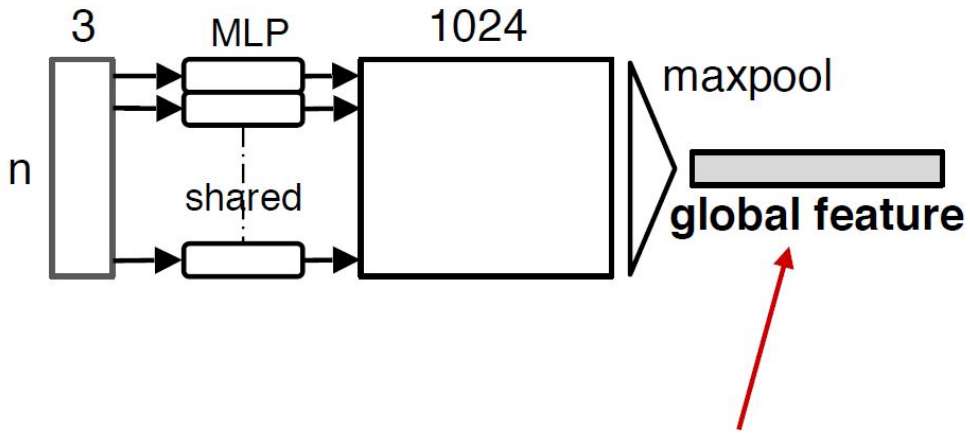


Image Source

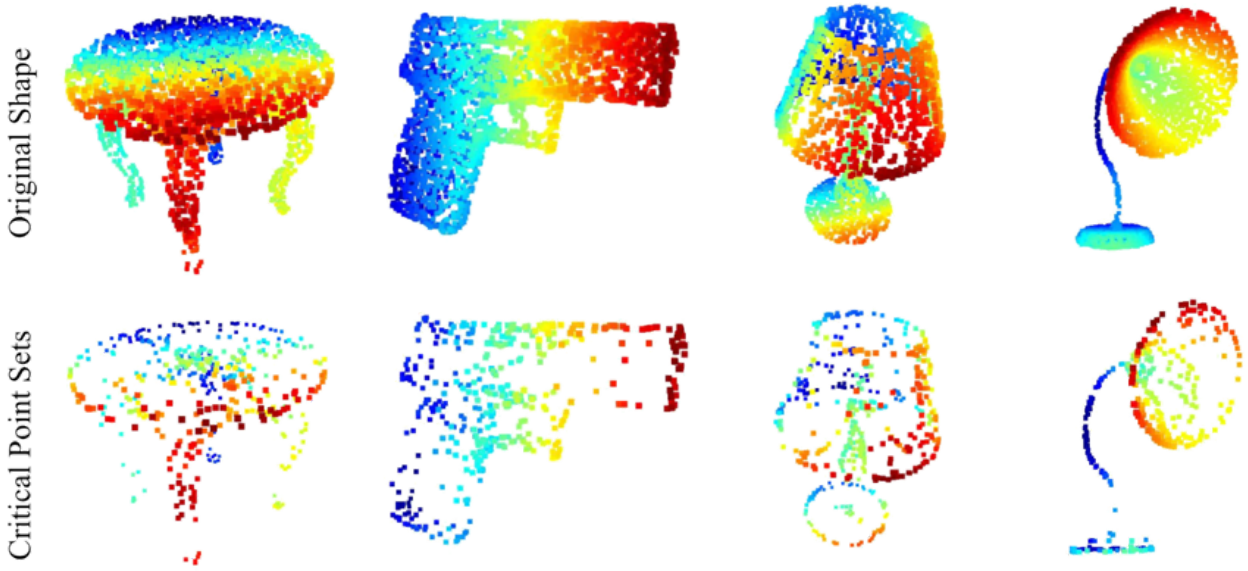
Visualizing Global Point Cloud Features



Which input points are contributing to the global feature?
(critical points)

Visualize What is Learned by Reconstruction

- Salient points are discovered!

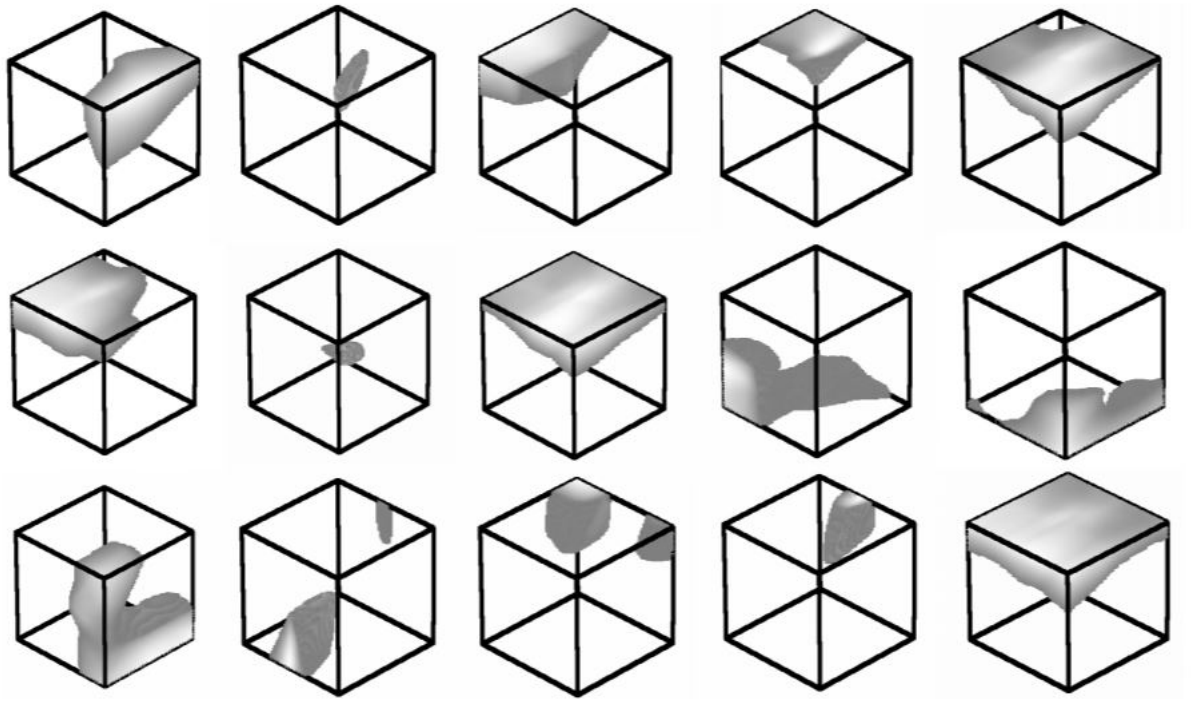


- The "critical points" are those who influenced the global feature vector, a.k.a the pooling layer.
  - The "critical" object's geometry is preserved.

Point function visualization

For each per-point function h (mlp), calculate the values of $h(p)$ for all the points p in the cube.

Random 15 functions out of the 1024 learned functions:



- Semi-equivalent to filter responses in CNNs

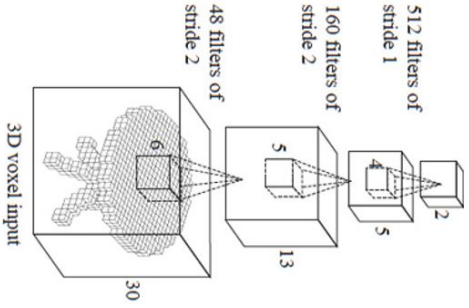
NA

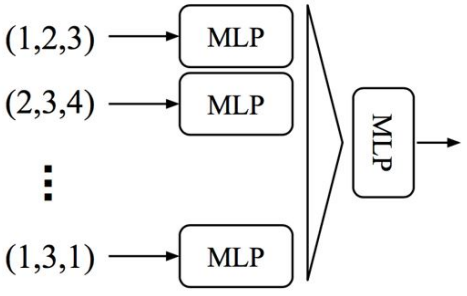
Limitations of PointNet

Hierarchical feature learning
Multiple levels of abstraction

v.s.

Global feature learning
Either one point or all points





3D CNN (Wu et al.)

PointNet (vanilla) (Qi et al.)

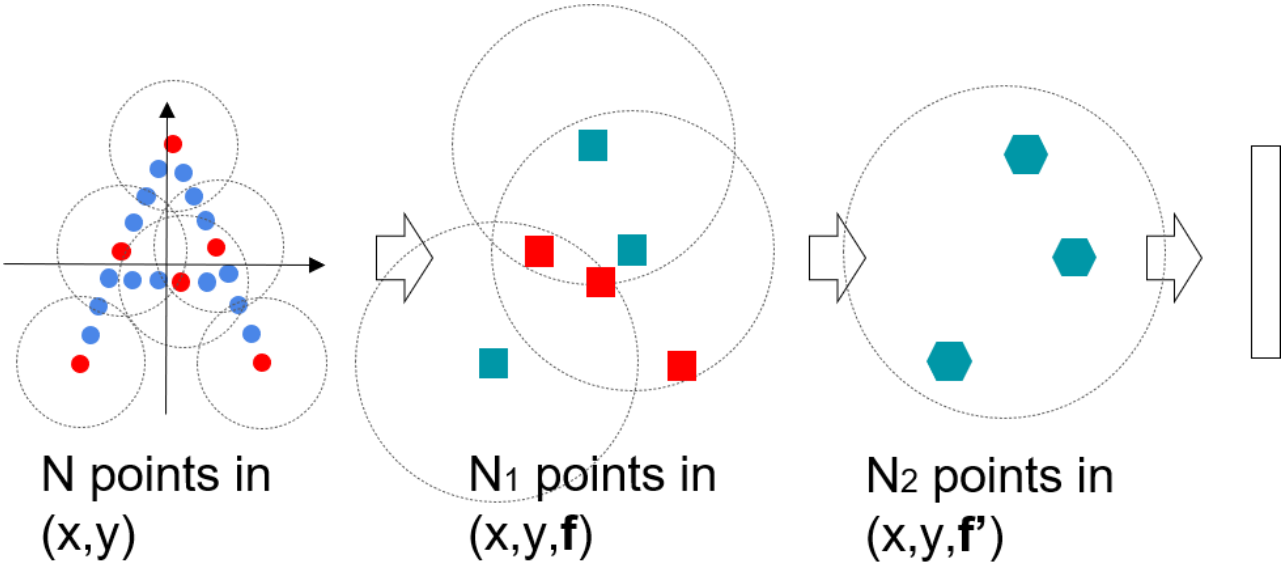
- No local context for each point
- Global feature depends on **absolute** coordinate. Hard to generalize to unseen scene configurations

Points in Metric Space

- Learn “kernels” in 3D space and conduct convolution
- Kernels have compact spatial support
- For convolution, we need to find neighboring points
- Possible strategies for range query
  - Ball query (results in more stable features)

- k-NN query (faster)

PointNet v2.0 (PointNet++): Multi-Scale PointNet



Repeated layers:

- Sample anchor points
- Find neighborhood of anchor points
- Apply PointNet in each neighborhood to mimic convolution

- Image Source

More Point Clouds DL solutions:

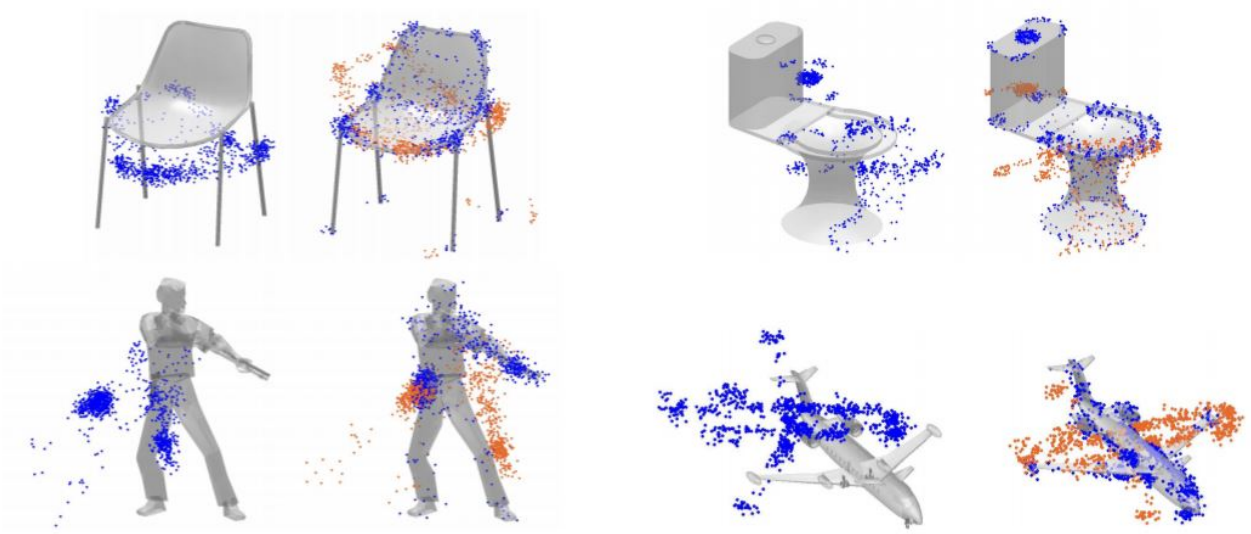
- 3DmFV
- Dynamic Graph CNN
- PCNN
- PointCNN
- KPConv

3D Deep Learning Applications

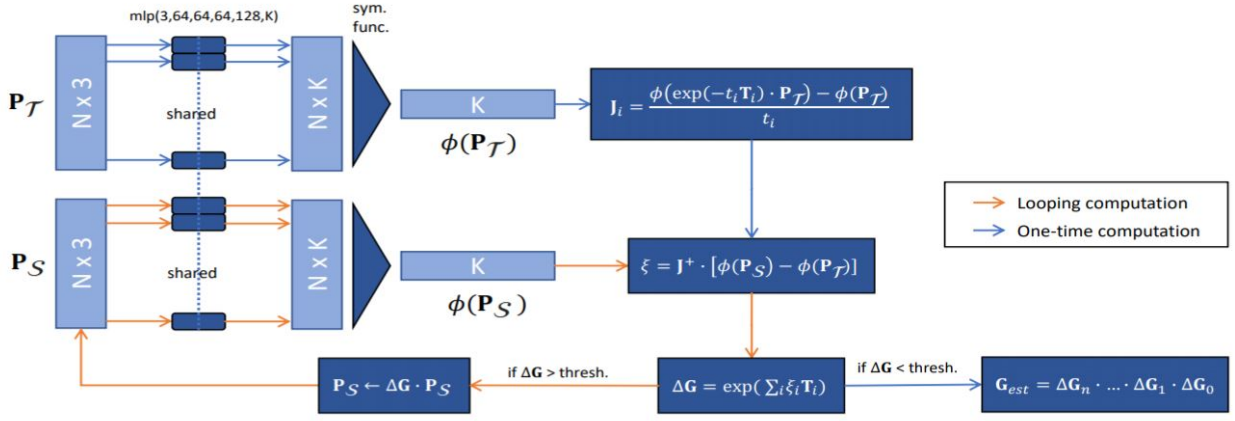
- Classification (V)
- Semantic segmentation (V)
- Part segmentation
- Registration (Upcoming)
- Generation (Upcoming)
- Object detection (Upcoming)
- Reconstruction
- Sampling - Downsampling, Upsampling
- SLAM
- Normal Estimation
- and many more...

Registration:

Problem statment: Find the rotation and translation transformation between objects



- PointNetLK (blue) - Deep Learning, based on Lucas–Kanade method (Tracking lecture)
  - Comparing 2 point clouds using PointNet features
- ICP (orange) - Classic registration method

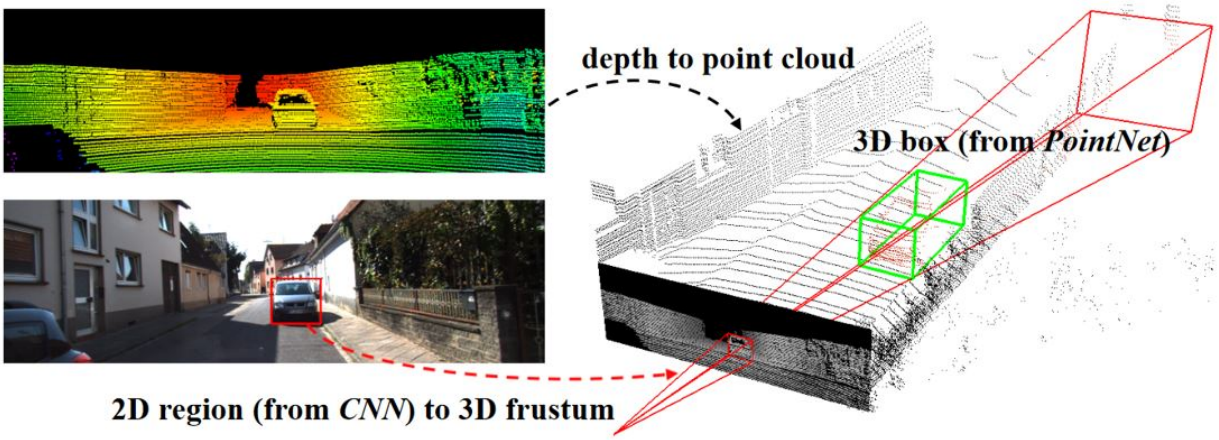


Both inputs (target and source) are being processed by PointNet architecture

Image Source

Detection:

- Generate object proposals from a view (e.g., using SSD)
- Recognize using PointNet



- Image Source

Questions

- What are the differences between 2D image and a point cloud?
  - Unstructured
  - Varying number of points
  - Unordered

Questions

- Why it might be hard to feed a point cloud as input to a neural network (NN)?
  - Does not rely on a grid
  - Does not have a fixed size
  - Different permutation represent the same point cloud

All three diffrences influence directly the ability of using NNs!

Questions

- What are the benefits of using a point cloud?
  - Most sensors raw outputs are point clouds (LiDAR)
  - Very efficient representation of 3D data (no empty voxels)
  - Reserve geometric details (no quantization)

Recommended Tools

Python:

- Open3D
- trimesh
- Ipyvolume - Visualization for Notebooks

Deep Learning:

- Pytorch3D
- Kaolin (Pytorch)
- TensorFlow Graphics

Recommended Tools

Visualize Tools (drop and view):

- CloudCompare
- MeshLab

Datasets:

- ModelNet
- ShapeNet
- PartNet
- Sydney Urban Opject DAtaset
- Stanford 3D
- KITTI
- ...

Recommended Tools

For more 3D deep learnig frameworks and datasets:

- awesome-point-cloud-analysis
- 3D-Machine-Learning#datasets

Recommended Videos

Warning!

- These videos do not replace the lectures and tutorials.
- Please use these to get a better understanding of the material, and not as an alternative to the written material.

Video By Subject

- 3D Deep Learning
  - General (Both highly recommended):
    - 3D Deep Learning Tutorial from SU lab at UCSD - Hao Su
    - Geometric deep learning - Michael Bronstein
  - PointNet
  - 3DmFV

Credits

- EE 046746 Spring 2021 - Elias Nehme
- EE 046746 Spring 2020 - Dahlia Urbach
- Slides - Yizhak (Itzik) Ben-Shabat, Simon Lucey (CMU), Hao Su, Jiayuan Gu and Minghua Liu(UCSanDiego)
- Multiple View Geometry in Computer Vision - Hartley and Zisserman - Sections 9,10
- Computer Vision: Algorithms and Applications - Richard Szeliski - Sections 11,12
- Research Papers:
    - PointNet - Qi et al. CVPR 17
    - PointNet++ - Qi et al. NeuroIPS 17
    - Generative Models for 3D Point Clouds - Qi et al. ICML 18
    - Frustum PointNets - Qi et al. CVPR 18
    - PointNetLTK - Aoki et al. CVPR 19
    - etc.
- Icons from Icon8.com - https://icons8.com