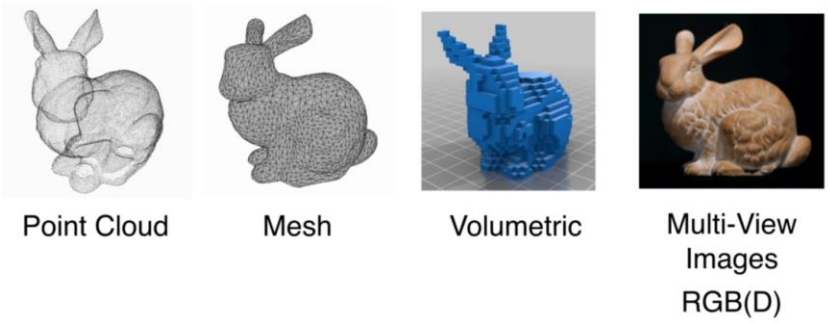
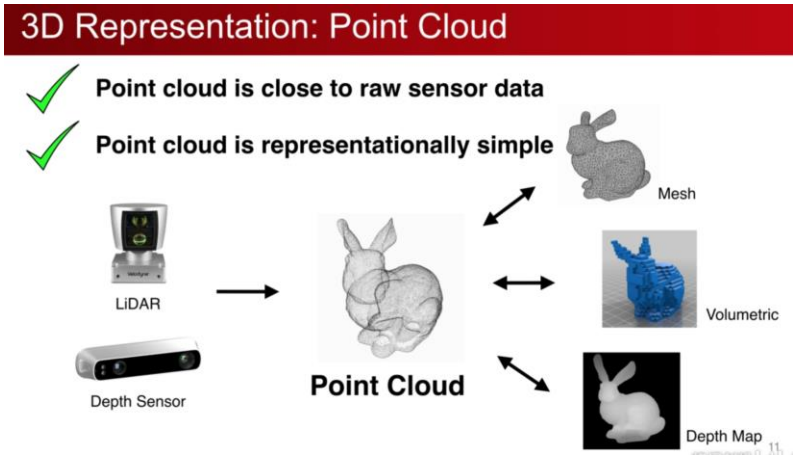
点云的深度学习

我们在自动驾驶中，要检测周围物体的类型和状态。

Require **data-driven method** to **process and understand 3D data**





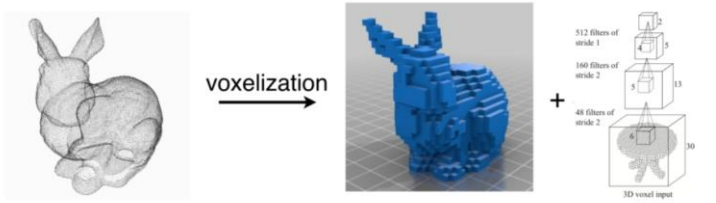
点云是适合于做 3D 表示形式的数据格式，这是因为激光雷达或者深度传感器，直接产生的其实就是点云的数据。而 mesh 则需要选择面片的形状，体素则需要选择分辨率，而分辨率的选择会影响表示的精度或速度，最后深度图像我们则需要选择多个角度才能完整表示一个物体。

Most existing point cloud features are **handcrafted** towards specific tasks

Feature Name	Supports Texture / Color	Local / Global / Regional	Best Use Case
PFH	No	L	
FPFH	No	L	2.5D Scans (Pseudo single position range images)
VFH	No	G	Object detection with basic pose estimation
CVFH	No	R	Object detection with basic pose estimation, detection of partial objects
RIFT	Yes	L	Real world 3D-Scans with no mirror effects. RIFT is vulnerable against flipping.

这些手动设计的特征都有其局限性，对于一个新的任务，我们很难用数据的方式来优化这些特征，我们希望进一步的往特征学习，深度学习的方向前进。

Point cloud is **converted to other representations** before it is input to a deep neural network



irregular point cloud

regular volumetric array

[Wu et al. 2015]

由于点云的不规则性，前期的点云深度学习都是先化为规则的表达方式，然后通过 3DCNN 来进行深度学习。但是 3DCNN 有很大的缺陷，3D 卷积的时间和空间复杂度随着分辨率三次幂的增长，计算代价很大，因此在实际中一般都采用很低的分辨率，分辨率低也会导致量化错误，导致噪声的出现，使得识别的准确率下降。如果我们不计复杂度去取高分辨率，会使得许多栅格都是空白的，我们只能扫描到表面，而内部都是空白的，这样的形式不是我们所乐见的。

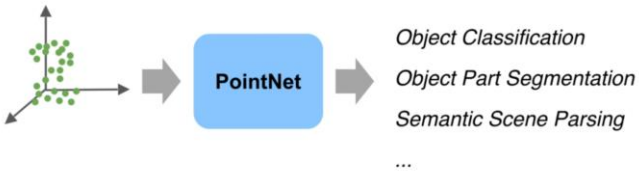
Point cloud is **converted to other representations** before it is input to a deep neural network

Conversion	Deep Net
Voxelization	3D CNN
Projection/Rendering	**2D CNN**
Feature extraction	Fully Connected

也有工作将点云投影或渲染到 2D 中，这样会导致 3D 信息的丢失，而且还要决定投影的角度，在很多时候是比较复杂的。还有工作是直接从点云中提取手工特征，然后将其进行全连接的操作，但是这样网络的性能被手工特征所局限。

End-to-end learning for irregular point data

Unified framework for various tasks

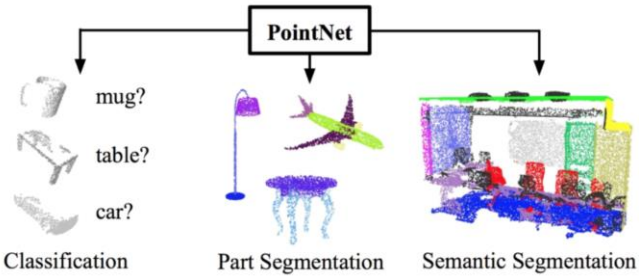


*[1] **Charles R. Qi**, Hao Su, Kaichun Mo, Leonidas J. Guibas. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. (CVPR'17)*

在 PointNet 中使用了一种端到端的直接基于点云的深度学习。

End-to-end learning for irregular point data

Unified framework for various tasks



使用 PointNet 可以实现分类、部分分割和语义分割等功能。

The model has to respect properties of point clouds:

Permutation Invariance

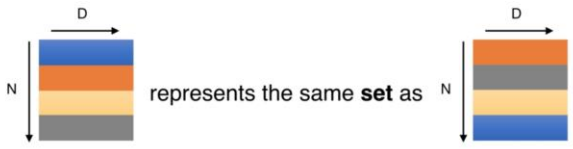
Point cloud is a set of unordered points

Transformation Invariance

Point cloud rotations should not alter classification results

点云具有无序性，或者说点云对排列并不敏感。

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



Model needs to be invariant to N! permutations

我们可以将点云数据简单的表示为一个二维的矩阵，其中 N 为点云的数量，D 为点云的维度，最简单的就是三维的 xyz，也可以是 xyzn 包含法线，也可以是 xyzrgb 包含颜色。总的来说，如果我们将矩阵的行随意呼唤，其实点云不会发生任何变化，这就要求网络对点云的所有置换都具有不变性，如果有 N 个点云数据，则有 N!种置换，网络就要确保其置换不变

性。在这里有一种系统的解决方案，即对称函数。

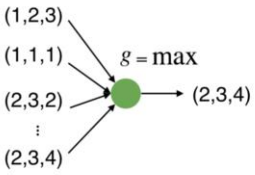
$$f(x_1,x_2,\ldots,x_n) \equiv f(x_{\pi_1},x_{\pi_2},\ldots,x_{\pi_n}), \quad x_i \in \mathbb{R}^D$$

Examples:

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

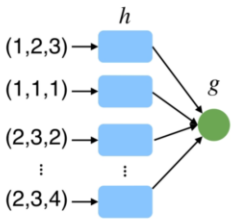
比如我们对 x 有一个 pi 的置换，置换完毕后仍然是完全相等的，这里比如说最大值函数和加法函数，都是典型的对称函数。

Simplest form: directly aggregate all points with a symmetric operator g
Just discovers very extreme/naive property of the geometry.



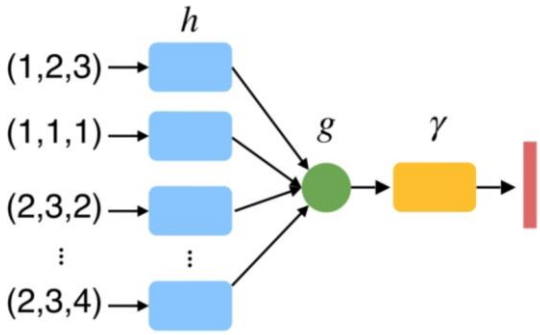
对点集做最大值或平均值的对称函数操作，我们会得到最大值或重心，但是这样会导致大部分点的丢失。

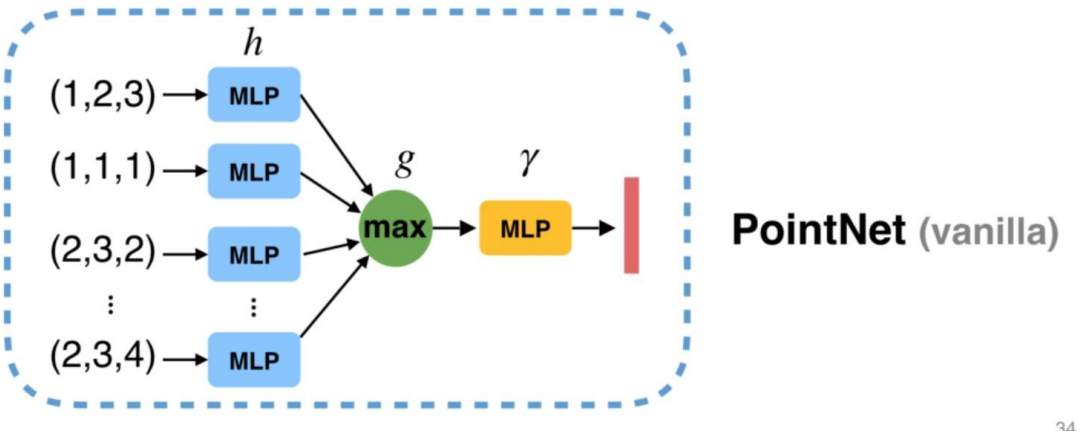
Embed points to a high-dim space before aggregation.
Aggregation in the (redundant) high-dim space preserves interesting properties of the geometry.



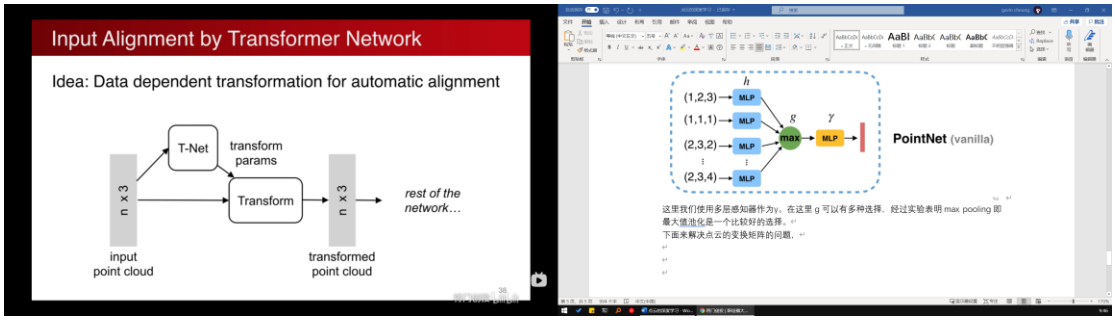
我们也可以在对称函数操作前进行高维表示，用高维空间表示 3 维的点肯定是冗余的，我们对高维的冗余信息进行对称性比如 max 操作，可以避免信息的丢失

$$f(x_1,x_2,\ldots,x_n) = \gamma \circ g(h(x_1),\ldots,h(x_n)) \text{ is symmetric if } g \text{ is symmetric}$$

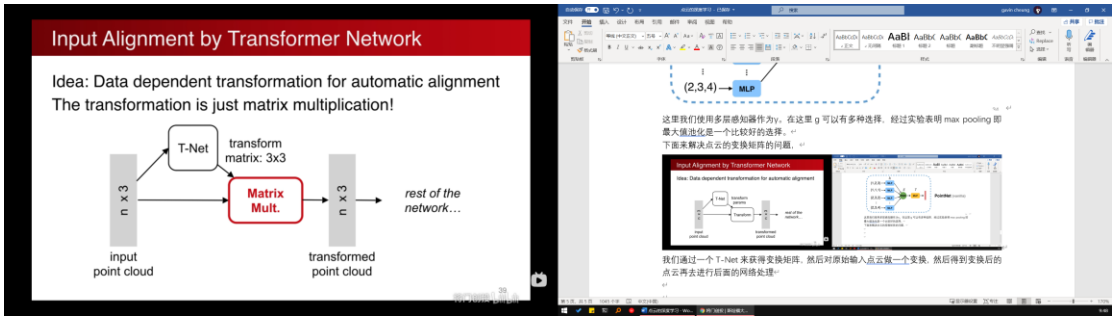




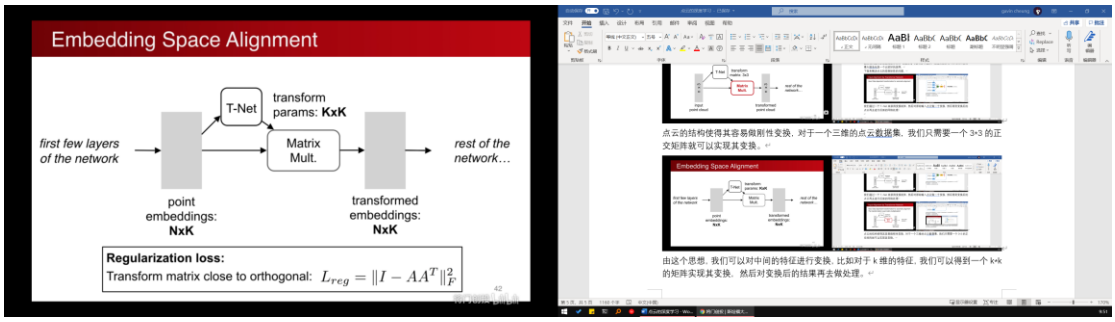
这里我们使用多层感知器作为γ。在这里 g 可以有多种选择，经过实验表明 max pooling 即最大值池化是一个比较好的选择。
下面来解决点云的变换矩阵的问题，



我们通过一个 T-Net 来获得变换矩阵，然后对原始输入点云做一个变换，然后得到变换后的点云再去进行后面的网络处理

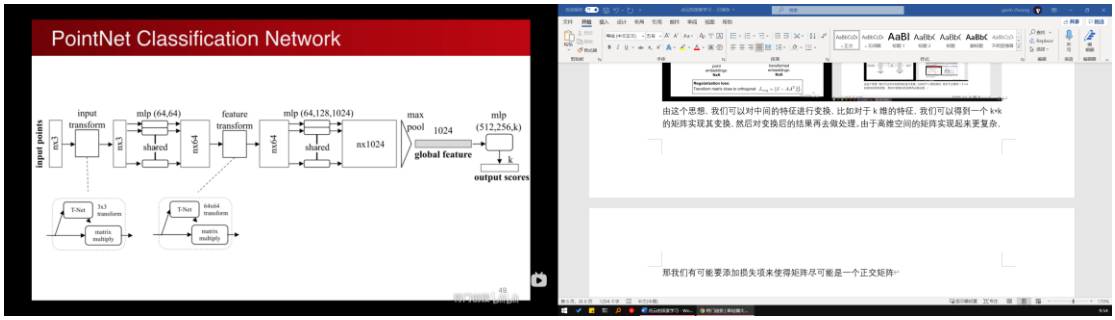


点云的结构使得其容易做刚性变换，对于一个三维的点云数据集，我们只需要一个 3*3 的正交矩阵就可以实现其变换。



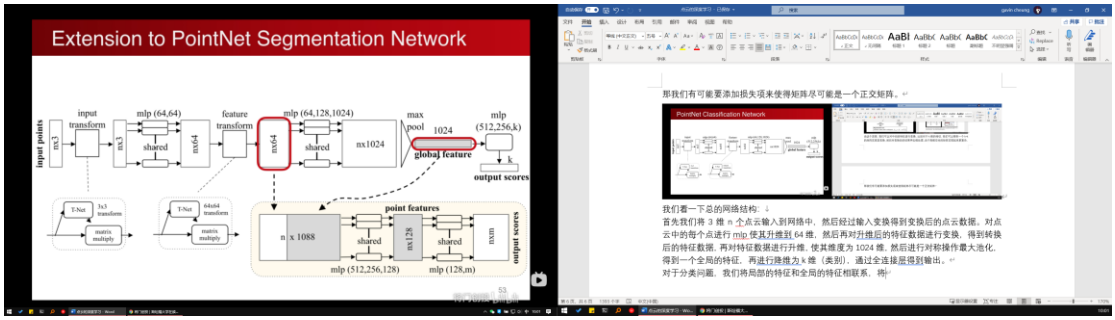
由这个思想，我们可以对中间的特征进行变换，比如对于 k 维的特征，我们可以得到一个 k*k 的矩阵实现其变换，然后对变换后的结果再去做处理。由于高维空间的矩阵实现起来更复杂，

那我们有可能要添加损失项来使得矩阵尽可能是一个正交矩阵。



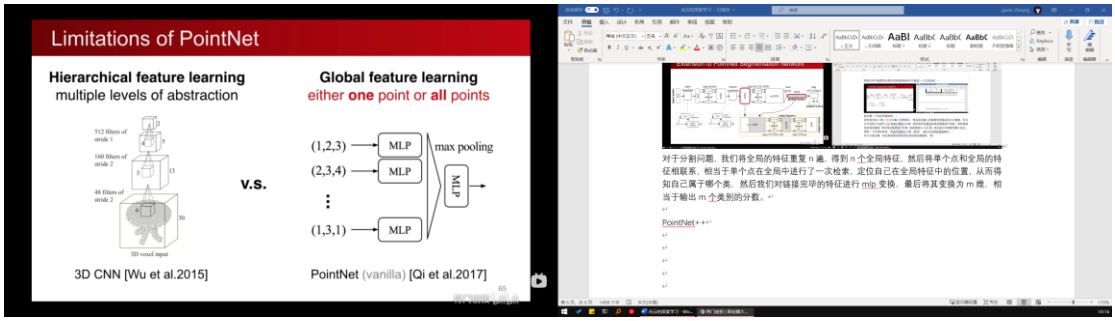
我们看一下总的网络结构：

首先我们将 3 维 n 个点云输入到网络中，然后经过输入变换得到变换后的点云数据。对点云中的每个点进行 mlp 使其升维到 64 维，然后再对升维后的特征数据进行变换，得到转换后的特征数据，再对特征数据进行升维，使其维度为 1024 维，然后进行对称操作最大池化，得到一个全局的特征，再进行降维为 k 维（类别），通过全连接层得到输出。

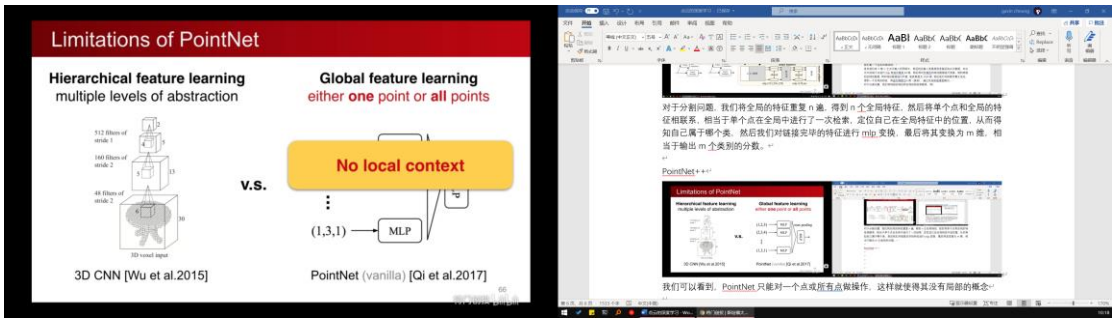


对于分割问题，我们将全局的特征重复 n 遍，得到 n 个全局特征，然后将单个点和全局的特征相联系，相当于单个点在全局中进行了一次检索，定位自己在全局特征中的位置，从而得知自己属于哪个类，然后我们对链接完毕的特征进行 mlp 变换，最后将其变换为 m 维，相当于输出 m 个类别的分数。

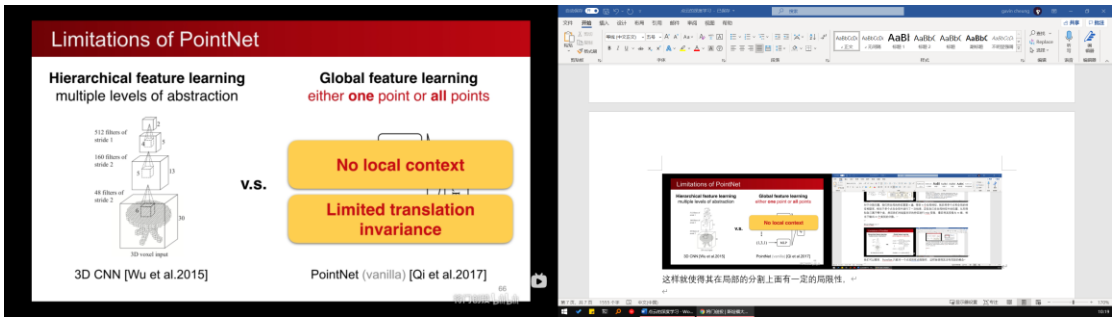
PointNet++



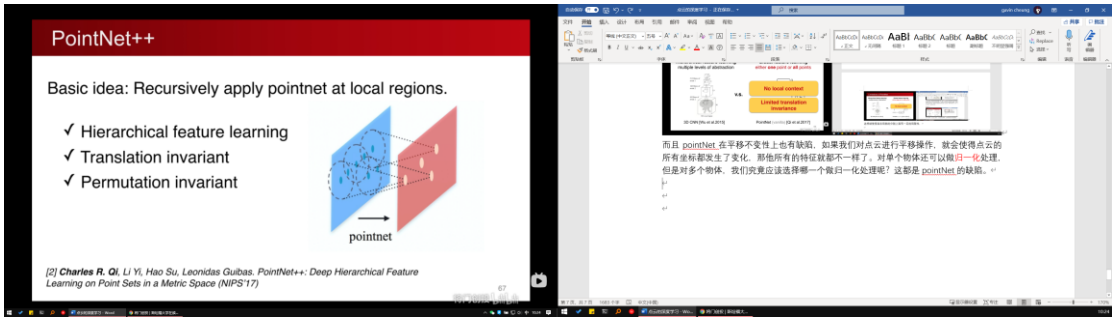
我们可以看到，PointNet 只能对一个点或所有点做操作，这样就使得其没有局部的概念



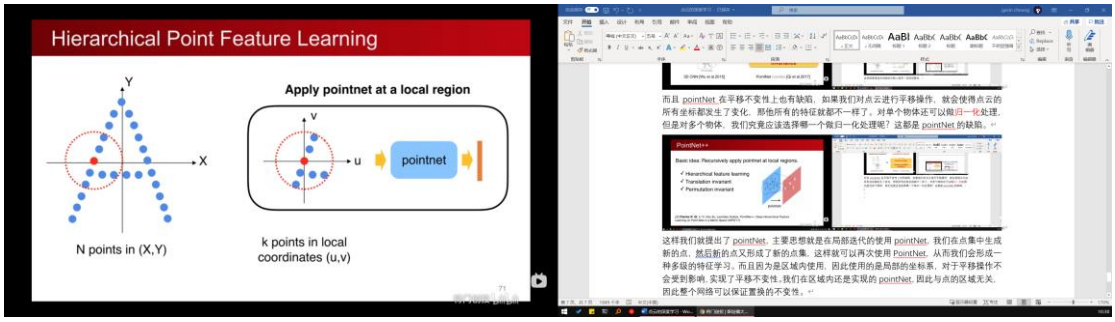
因此很难在很精细的特征中进行学习，这样就使得其在局部的分割上面有一定的局限性。



而且 pointNet 在平移不变性上也有缺陷，如果我们对点云进行平移操作，就会使得点云的所有坐标都发生了变化，那他所有的特征就都不一样了。对单个物体还可以做归一化处理，但是对多个物体，我们究竟应该选择哪一个做归一化处理呢？这都是 pointNet 的缺陷。

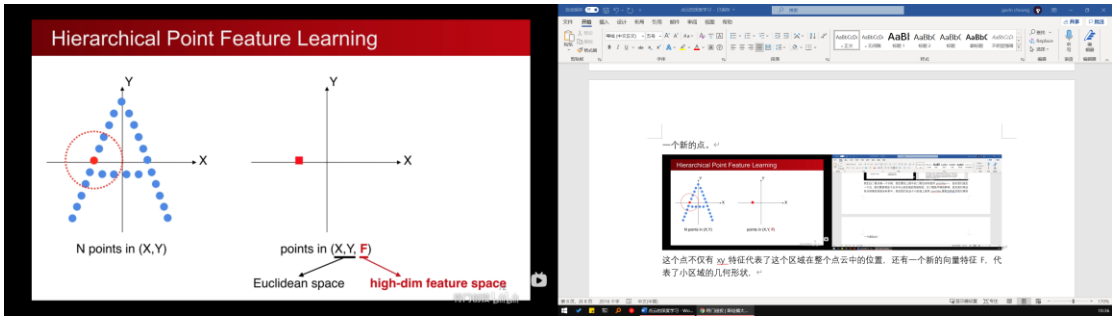


这样我们就提出了 pointNet，主要思想就是在局部迭代的使用 pointNet，我们在点集中生成新的点，然后新的点又形成了新的点集，这样就可以再次使用 PointNet，从而我们会形成一种多级的特征学习。而且因为是区域内使用，因此使用的是局部的坐标系，对于平移操作不会受到影响，实现了平移不变性。我们在区域内还是实现的 pointNet，因此与点的区域无关，因此整个网络可以保证置换的不变性。

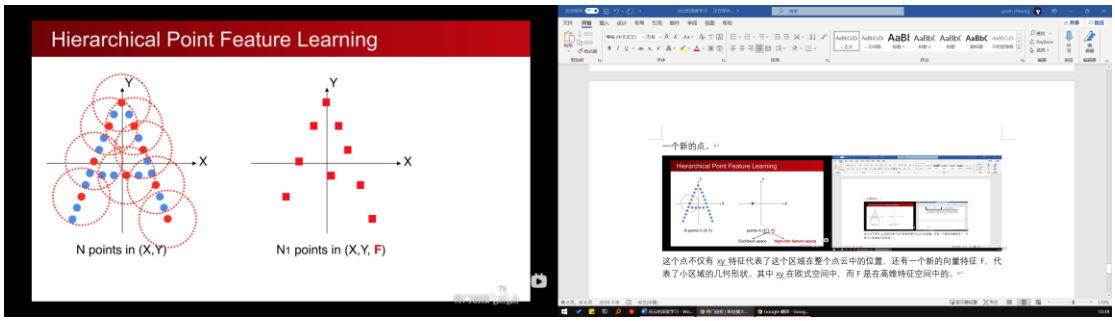


我们以二维点做一个示例，我们要在上图中的二维空间内使用 pointNet++，首先我们选定一个点，我们要获得这个点为中心的区域的局部特征。为了避免平移的影响，首先我们将这些点转换到局部坐标系中。然后我们在这个小区域上使用 pointNet,提取完特征后我们得到

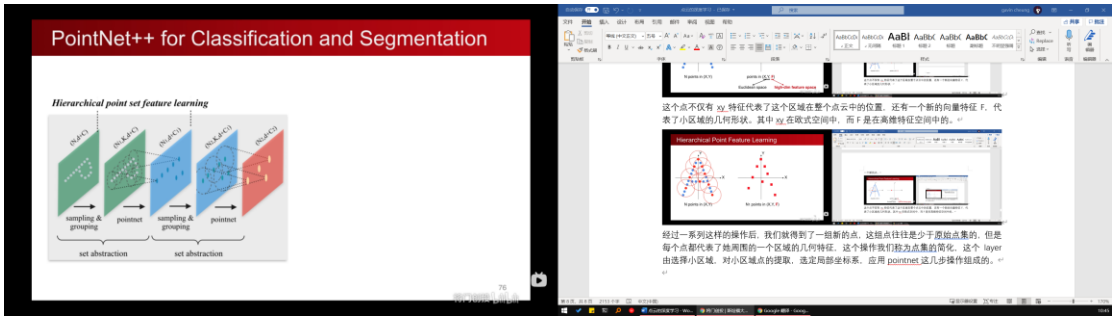
一个新的点。



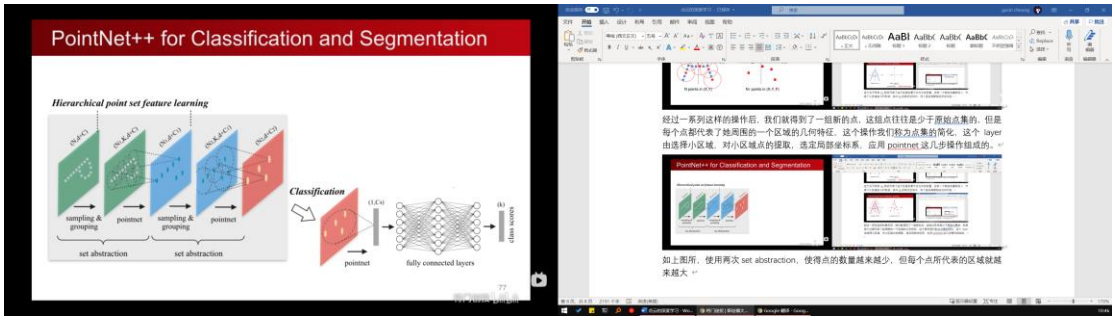
这个点不仅有 xy 特征代表了这个区域在整个点云中的位置，还有一个新的向量特征 F，代表了小区域的几何形状。其中 xy 在欧式空间中，而 F 是在高维特征空间中的。



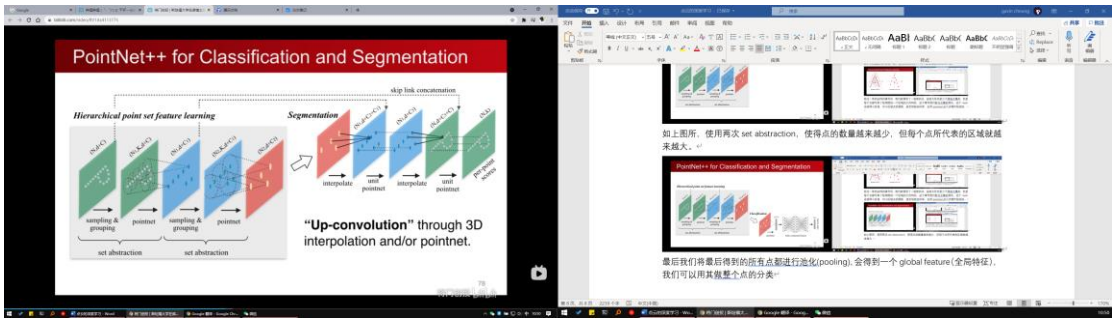
经过一系列这样的操作后，我们就得到了一组新的点，这组点往往是少于原始点集的，但是每个点都代表了她周围的一个区域的几何特征，这个操作我们称为点集的简化，这个 layer 由选择小区域，对小区域点的提取，选定局部坐标系，应用 pointnet 这几步操作组成的。



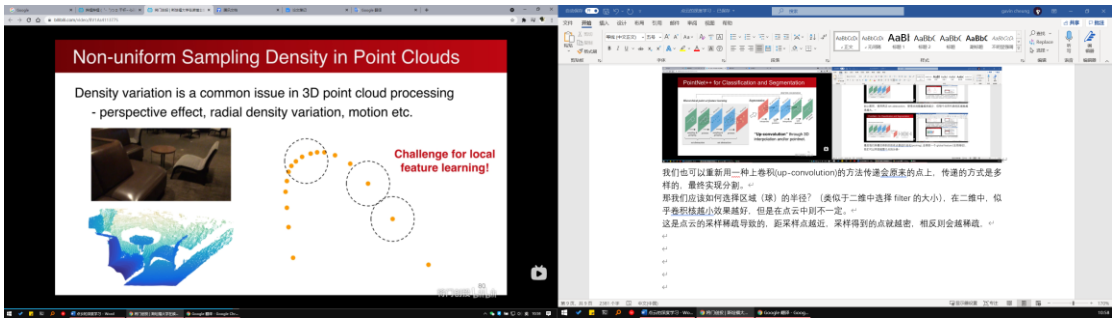
如上图所，使用两次 set abstraction，使得点的数量越来越少，但每个点所代表的区域就越来越大。



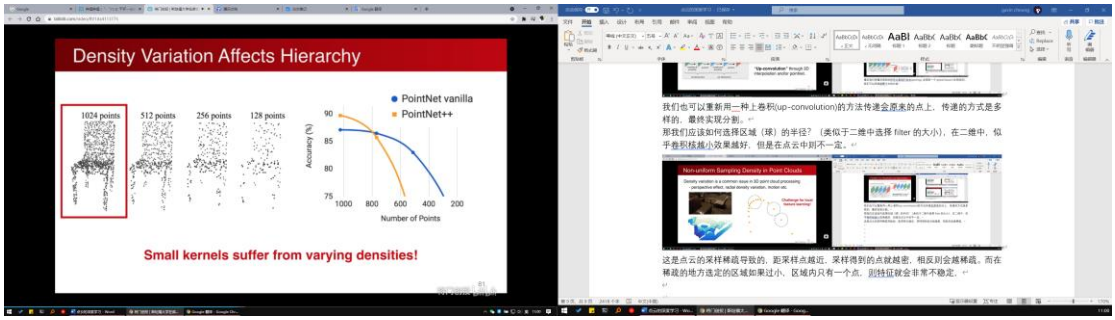
最后我们将最后得到的所有点都进行池化(pooling)，会得到一个 global feature（全局特征），我们可以用其做整个点的分类



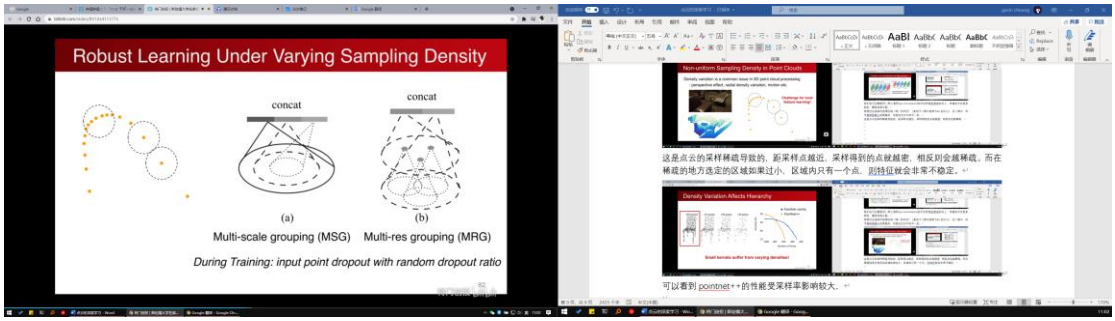
我们也可以重新用一种上卷积(up-convolution)的方法传递会原来的点上，传递的方式是多样的，最终实现分割。
那我们应该如何选择区域（球）的半径？（类似于二维中选择 filter 的大小），在二维中，似乎卷积核越小效果越好，但是在点云中则不一定。



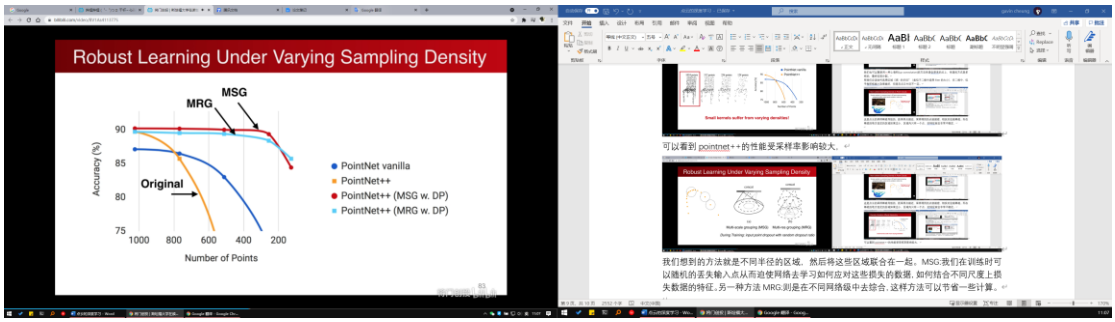
这是点云的采样稀疏导致的，距采样点越近，采样得到的点就越密，相反则会越稀疏。而在稀疏的地方选定的区域如果过小，区域内只有一个点，则特征就会非常不稳定。



可以看到 pointnet++的性能受采样率影响较大，



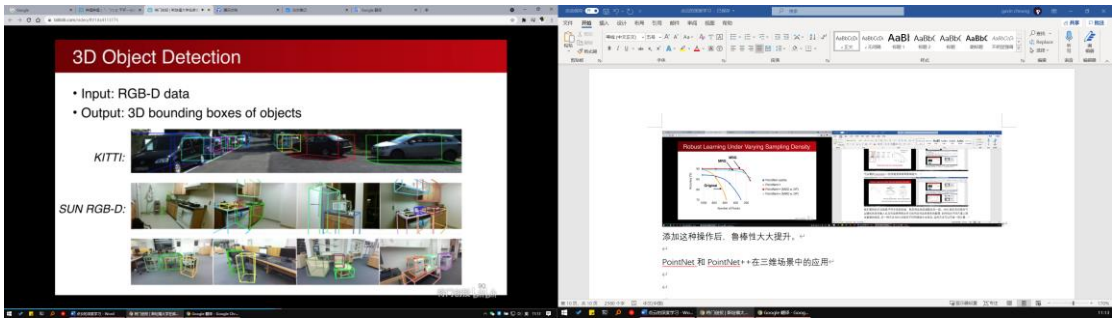
我们想到的方法就是不同半径的区域，然后将这些区域联合在一起。MSG:我们在训练时可以随机的丢失输入点从而迫使网络去学习如何应对这些损失的数据，如何结合不同尺度上损失数据的特征。另一种方法 MRG:则是在不同网络级中去综合，这样方法可以节省一些计算。

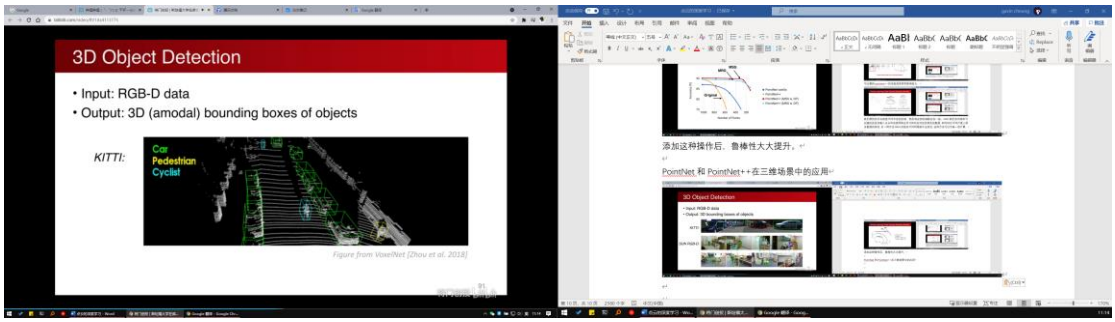


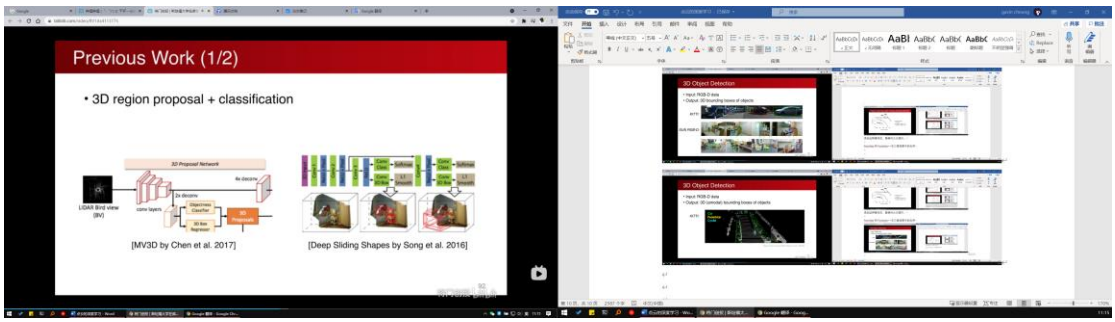
添加这种操作后，鲁棒性大大提升。

PointNet 和 PointNet++在三维场景中的应用

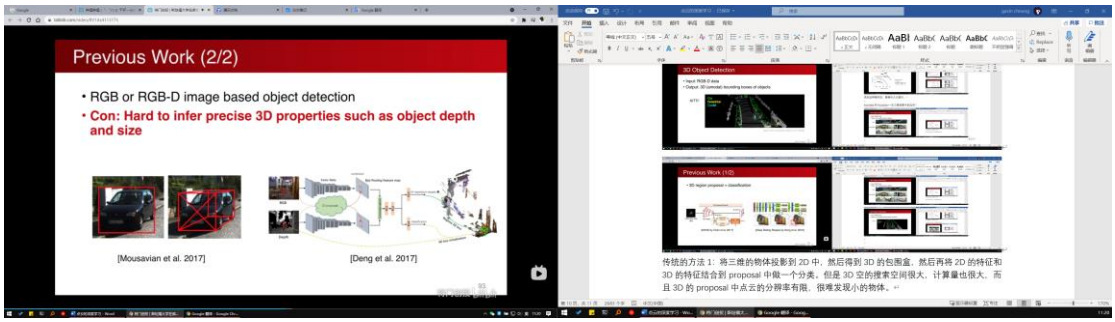
三维物体的识别



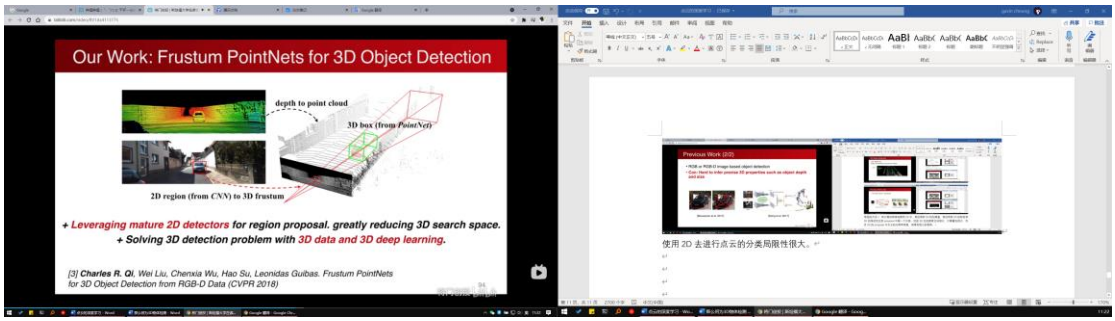




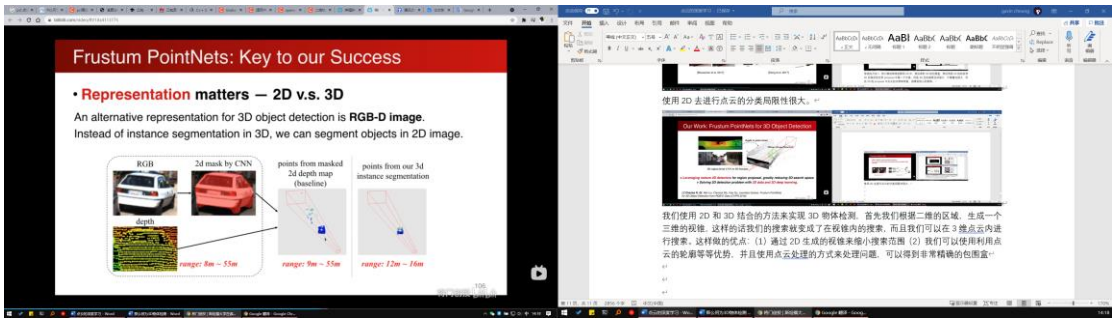
传统的方法 1：将三维的物体投影到 2D 中，然后得到 3D 的包围盒，然后再将 2D 的特征和 3D 的特征结合到 proposal 中做一个分类。但是 3D 空的搜索空间很大，计算量也很大，而且 3D 的 proposal 中点云的分辨率有限，很难发现小的物体。



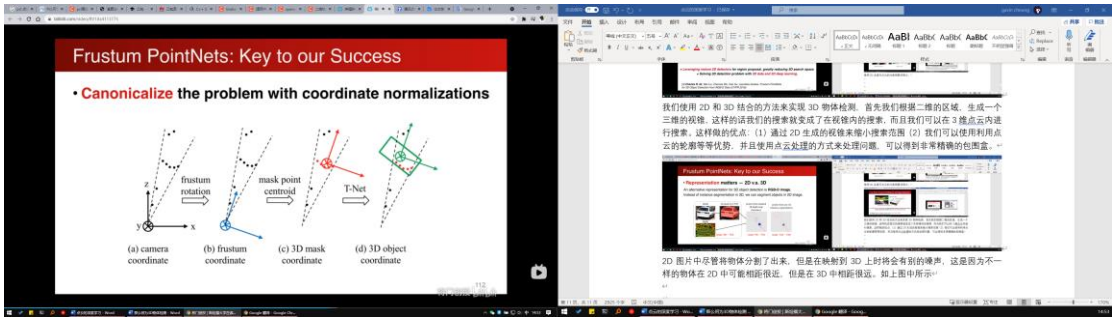
使用 2D 去进行点云的分类局限性很大。



我们使用 2D 和 3D 结合的方法来实现 3D 物体检测，首先我们根据二维的区域，生成一个三维的视锥，这样的话我们的搜索就变成了在视锥内的搜索，而且我们可以在 3 维点云内进行搜索。这样做的优点：(1) 通过 2D 生成的视锥来缩小搜索范围 (2) 我们可以使用利用点云的轮廓等等优势，并且使用点云处理的方式来处理问题，可以得到非常精确的包围盒。



2D 图片中尽管将物体分割了出来，但是在映射到 3D 上时将会有别的噪声，这是因为不一样的物体在 2D 中可能相距很近，但是在 3D 中相距很远。如上图中所示



归一化操作：由于是点云的数据类型，因此归一化非常简单，只需要一些矩阵的旋转操作即可