Chapter 11
Deep Learning on 3D Data

Charles Ruizhongtai Qi

Abstract Emerging 3D related applications such as autonomous vehicles, AI-assisted design, and augmented reality have highlighted the demands for more robust and powerful 3D analyzing algorithms. Inspired by the success of deep learning on understanding images, audio, and texts, and backed by growing amounts of available 3D data and annotated 3D datasets, a new field that studies deep learning on 3D data has arisen recently. However, unlike images or audio that have a dominant representation as arrays, 3D has many popular representations. Among them, the two most common representations are point clouds (from raw sensor input) and meshes (widely used in shape modeling) that are both not defined on a regular grid. Due to their irregular format, current convolutional deep neural networks cannot be directly used. To analyze those 3D data, two major branches of methods exist. One family of methods first converts such irregular data to regular structures such as 3D volumetric grids (through quantization) or multi-view images (through rendering or projection) and then applies existing convolutional architectures on them. On the other hand, a new family of methods study how to design deep neural networks that *directly* consume irregular data such as point clouds (sets) and meshes (graphs). Those architectures are designed to respect the special properties of the input 3D representations such as the permutation invariance of the points in a set, or the intrinsic surface structure in a mesh. In this chapter we present representative deep learning models from both of those families, to analyze 3D data in representations of regular structures (multi-view images and volumetric grids) and irregular structures (point clouds and meshes). While we mainly focus on introducing the backbone networks that are general for deep 3D representation learning, we also show their successful applications ranging from semantic object classification, object part segmentation, scene parsing, to finding shape correspondences. At the end of the chapter we provide more pointers for further reading and discuss future directions.

C. Ruizhongtai Qi (✉)
Stanford University, Stanford, CA, USA
e-mail: charlesq34@gmail.com

Y. Liu et al. (eds.), *3D Imaging, Analysis and Applications*,
https://doi.org/10.1007/978-3-030-44070-1_11

11.1 Introduction

Deep learning is part of a broader family of machine learning methods, while machine learning itself belongs to the even broader field of artificial intelligence. Compared to designing handcrafted features, deep learning allows computational models with multiple processing layers to *learn representations* of data with multiple levels of abstraction [54]. Artificial neural networks are the most popular realizations of deep learning models. In recent years, the growing computing power provided by GPUs (graphics processing units) and the availability of large-scale training datasets have fueled the fast development of deep learning models and algorithms. This has led to breakthroughs in many areas of computer science, from speech recognition [2, 66, 101] to image understanding [37, 52, 79, 88], to machine translation [108]. Methods based on deep learning have even achieved super-human performance on many problems that were previously considered very hard for machines. One example is a computer Go program called AlphaGo [87] that beat multiple top human players, where deep learning plays a key role in its system. Nowadays, deep learning models have already been widely used in daily products, such as Siri and Face ID on iPhones, Google Image Search and Google Translate, and the Autopilot system in Tesla cars for semi-autonomous driving.

While deep learning has great success in processing data such as image, videos, audio, and texts, only until very recently researchers started to explore how to learn deep representations from 3D data such as point clouds and meshes, leading to a rising field named *3D deep learning*. The task space of 3D deep learning is vast but can be roughly divided into three categories: 3D analysis, 3D synthesis, and 3D assisted image understanding [91]. In 3D geometry analysis, some typical tasks include semantic classification of objects [65, 76, 92, 110] (given a 3D object, classify it into one of a few predefined categories such as chairs and tables), semantic parsing of objects or scenes [3, 19, 77] (segmenting an object or a scene into different semantic regions such as chair leg, chair back for a chair object, or floor, wall, table for a scan of an indoor room), and finding correspondences among different shapes [9] (such as human bodies). In 3D synthesis, a deep learning model generates 3D data either from conditioned input or an embedding vector. For example, one interesting problem is shape completion from partial scans, where given a partial scan we want to use a deep learning model to complete the scan based on its learned object shape priors [20]. Another example is automated shape modeling, where a model generates a mesh or parametric model of an object from unstructured input such as images and point clouds [100]. The third large category of tasks, 3D assisted image understanding, does not directly consume 3D data but uses 3D as a faithful surrogate to assist image understanding tasks [56, 86] that is hard to achieve without 3D data. For example, for the problem of intrinsic decomposition, we can generate synthetic data from existing 3D models to train our network.

Among those tasks this chapter focuses on 3D analysis tasks. Furthermore, we will focus on the "backbone" network architectures instead of discussing specific problems. These backbone architectures are the keys to deep 3D representation learning

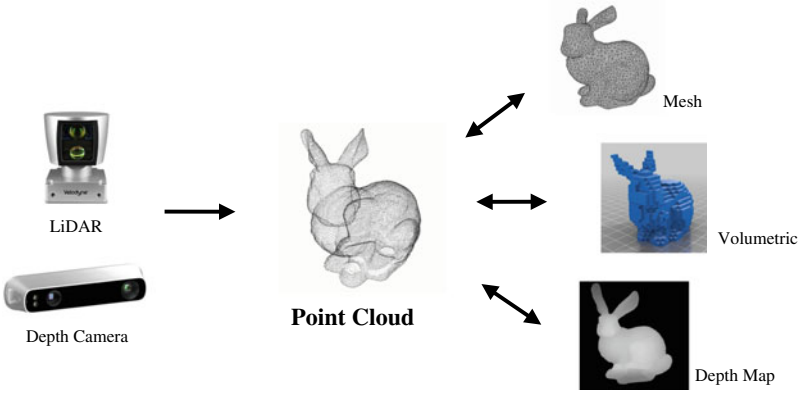


Fig. 11.1 3D representations. Among them, point clouds and meshes are in an irregular structure while depth maps (or projected views) and volumetric grids are in a regular structure that can be directly consumed by convolutional neural networks. Figure is from [74]

and are, therefore, the foundation for any applications built on top of 3D data. We will see that even just discussing about backbone architectures, we have a large space to explore and many challenges to solve.

One of the major challenges to develop deep learning models for 3D data is its representation issue. Unlike images that have a dominant representation as 2D pixel arrays, 3D has many popular representations, as shown in Fig. 11.1. In choosing or designing a deep learning model we need to first determine which 3D representation to use. The diversity of representations exists for a reason though: just as computational algorithms depend on data structures (lists, trees, graphs, etc.), different applications of 3D data have their own preferences for 3D representations. Among them, a point cloud is a set of points in space sampled from object surfaces, usually collected by 3D sensors such as LiDARs (Light Detection and Ranging) or depth cameras. A polygon mesh is a collection of triangles or quads and is heavily used in computer graphics because it is friendly to 3D modeling and rendering algorithms. Volumetric representation quantifies the space into small voxels (3D cubes) and is common in simulation and reconstruction (such as in medical imaging), because it is easier to aggregate signals in regular grids than other formats. Last but not least, we also represent 3D as multiple projected views, often used for visualization, as humans are more used to perceiving 2D than 3D. There are also more parametric representations such as those defined with cylinders, cubes, and NURBS, which are convenient to deform and simple to encode.

Among the above mentioned 3D representations the two most common ones are point clouds (from sensor data) and meshes (from geometric modeling). However, neither of them are defined on regular grids. Due to their irregularity, deep convolutional neural networks cannot be directly applied here. To analyze them, two branches of methods exist. One is to first convert such irregular data to a regular grid structure, such as volumetric grids (through voxelization) or images (through

rendering or projection) and then apply convolutional networks. The other approach is more end-to-end learning, where point clouds or meshes are directly inputted to the networks without any conversion. This however, requires new types of deep neural networks that respect the characteristics of the input 3D representations, such as the permutation invariance of points in a point set, or the intrinsic surface structure of a mesh. In this chapter, we present a few relatively mature and representative deep architectures for both regular and irregular 3D data. Particularly, in Sect. 11.3 we show deep learning models on regularly structured 3D representations (multi-view images and volumetric grids), with Multi-view CNN [92] and VoxNet [65] as two example architectures. In Sect. 11.4, we present deep learning models on point clouds with PointNet [76]. In Sect. 11.5, we briefly introduce deep learning methods on processing meshes by explaining motivations behind two representative works SyncSpecCNN [115] and ACNN [9]. Lastly we conclude the chapter, discuss future research directions in Sect. 11.6 and point readers to more resources for further reading in Sect. 11.7. In the end of the chapter, there are questions and exercises that provide an opportunity to check the reader's understanding of this chapter.

After reading this chapter, the readers should understand the challenges of deep learning on 3D data, be able to describe and implement a deep learning model for multi-view images, be able to describe and implement volumetric CNNs, be able to describe and implement deep learning models for processing point clouds, know about the two common types of deep networks on meshes, and be aware of future research directions in 3D deep learning.

11.2 Background

Recent advances in deep learning are supported by high-performance computing hardware (GPU) and new software frameworks (Caffe [46], TensorFlow [1], PyTorch [71], etc.), but more importantly they are fueled by large-scale datasets (e.g., ImageNet [23]). Similarly, 3D deep learning rises because *3D data* and *3D datasets* become available.

The growth in 3D data is due to two forces. The first driving force is the progress in 3D sensing technology. As the costs of sensors decrease quickly, we now have access to very affordable depth cameras such as Kinect, Tango, Structure Sensor, Intel RealSense or even depth cameras in phones (e.g., iPhone X). Extremely accurate 3D sensors such as LiDARs are also more widespread thanks to the needs of the autonomous vehicle industry. A large number of 3D sensors directly result in much quicker growth of raw 3D data. The second force is the availability and popularity of free 3D modeling tools such as Blender and SketchUp, as well as 3D model sharing platforms such as the Trimble 3D Warehouse, leading to a fast growing number of 3D models that are publicly accessible.

However, while 3D data quickly grows from both sensing and modeling, they are not yet ready to be used by deep learning algorithms, because most models still require annotated data for training. Fortunately, we also observe a few recent efforts

in organizing, cleaning, and annotating 3D data (e.g., ShapeNet [14]). The works in this chapter are only possible with these public 3D datasets.

In Sect. 11.2.1, we introduce the datasets we used in this chapter and discuss their characteristics. Then in Sect. 11.2.2, we survey previous work and also discuss recent advances in the field.

11.2.1 Datasets

We used various datasets for a variety of 3D analysis problems to test the performance of the presented deep architectures and algorithms. The datasets involved are diverse, ranging from single objects to multi-objects in scenes, from synthetic shapes to real scans.

- ModelNet40 [110]: A 3D dataset of CAD models from 40 categories (mostly man-made objects such as chairs, bathtubs, cars, etc.). All models are aligned in upright poses. In default, 10 of the 40 categories have aligned poses in azimuth angles. There is also a recent update in the dataset, providing fully aligned models for all categories. We use the official split with 9,843 shapes for training and 2,468 for testing to evaluate deep models in 3D shape classification, with random shape rotations in the azimuth direction (along the up-down axis).
- ShapeNetPart [114]: This is a dataset built on top of the ShapeNet [14] dataset (a large-scale, richly annotated 3D shape repository), containing 16,881 shapes from 16 of the ShapeNet classes, annotated with 50 parts in total. We use the official train/test split following [14]. We test deep models for object part segmentation on this dataset, which is a much finer grained task than whole shape classification in ModelNet40.
- S3DIS [3]: The Stanford 3D Indoor Spaces (S3DIS) dataset contains 3D scans from Matterport scanners in 6 building areas including 271 rooms. Each point in the scan is annotated as one of the 13 semantic categories (chair, table, floor, wall, etc., plus clutter). The dataset also provides annotations for depth and normal maps, as well as instance segmentations. We use this dataset for the evaluation of semantic segmentation (semantic scene parsing).

Besides the datasets mentioned above (those used in experiments in this chapter), there are many more other efforts in building 3D datasets. Examples include the SHREC15 [58] dataset with nonrigid objects, ScanNet [19] and Matterport3D [13] of large-scale indoor scans, as well as datasets with LiDAR scans such as KITTI [29] and Semantic3D [36]. There are also ongoing efforts to build large 3D/RGB-D video datasets and simulation platforms [85] for 3D understanding and interactions. We foresee that datasets will keep playing an important role in future 3D deep learning research.

11.2.2 Related Work

As 3D is prevalent in computer vision, graphics, and robotics, there have been tremendous works on topics of point clouds, 3D data learning, and 3D scene understanding. In this section, we briefly review previous works related to the topic, and introduce some concurrent and more recent efforts since the draft of this chapter. We first review prior attempts in designing and learning features for point clouds, then discuss 3D deep learning on point clouds and meshes.

11.2.2.1 3D Descriptors

Most existing features for point clouds and meshes are handcrafted toward specific tasks. These features or 3D descriptors often encode certain statistical properties of local points or surfaces and are designed to be invariant to certain transformations, which are typically classified as intrinsic (WKS [4], HKS [10, 94], etc.) or extrinsic (PFH [81], FPFH [82], D2 [69], inner-distance [60], Spin Image [47], LFD [15], etc.). They can also be categorized as local features and global features, or be classified based on the input format they take (e.g., 3D points, points with features, or depth maps). For a specific task, it is not trivial to find the optimal feature combination (feature selection), even with a deep neural network [28].

In this chapter we present deep learning approaches for 3D representation learning, which can be easily adapted to different tasks, such as classification, segmentation, shape retrieval, and finding correspondences.

11.2.2.2 Deep Learning on Point Clouds

Vinyals et al. [103] is one of the first researchers that studied deep learning on point sets. They designed a read-process-write network with attention mechanism to consume unordered input sets and showed that their network has the ability to sort numbers. However, since their work focuses on generic sets and natural language processing (NLP) applications, there lacks the role of geometry in the sets. Concurrent to the PointNet [76] we will introduce in this chapter, there are also works from Ravanbakhshet et al. [78] and Zaheer et al. [116], in which they designed deep networks that achieve invariance to set element ordering. However, their works emphasize a wide range of applications for point set learning, rather than focusing on 3D understanding. Compared with these prior work, the architecture we present in Sect. 11.4 exploits the geometry properties in 3D point sets (geometric transformations and distance spaces for sampling and local context). We also provide theoretical analysis and intuitive visualization of what has been learned by the deep network.

Real 3D point clouds from sensors or geometric estimation (e.g., structure from motion) are usually noisy and have nonuniform sampling densities. This affects effective point feature extraction and causes difficulty for learning. One of the key issues

is to select a proper scale for point feature design. Previously several approaches have been developed regarding this [6, 22, 33, 67, 72, 107] either in the geometry processing community or photogrammetry and remote sensing community. In none of the above deep learning works, the problem of nonuniform sampling density has been explicitly considered. In a recent work [77], a new architecture called PointNet++ was proposed which learns to extract point features and balance multiple feature scales in an end-to-end fashion. The PointNet++ has a hierarchical structure that applies shared subnetworks (based on the PointNet [76] architecture) at local regions of point clouds (similar to shared convolution kernels in a CNN) such that it is translation invariant. The hierarchical architecture shows stronger generalizability to large-scale 3D scene parsing than [76]. The nonuniform density problem has also been addressed more thoroughly in [39].

Recently there have been many more efforts in designing novel deep neural networks for point clouds [16, 32, 39, 44, 51, 55, 57, 80, 93, 97, 98, 104–106, 111, 113, 117]. For example, dynamic graph CNNs [105] extends PointNet++ by allowing neighborhood search in latent feature spaces. VoxelNet [117] combines PointNet with 3D CNNs. It computes local voxel features using a PointNet-like network, and then applies 3D CNNs on voxels for object proposals and classification. O-CNN [106], OctNet [80] and Kd-network [51] have introduced indexing trees to 3D deep learning, to avoid computations at empty spaces. ShapeContextNet [111] extends the traditional Shape Context [5] descriptor to a trainable setting with deep neural networks. SPLATNet [93] achieves efficient 3D convolution by sparse bilateral convolutions on a lattice structure. Tangent convolution network [97] exploits the fact that point clouds from 3D sensors are living on a 2D manifold in 3D space, and achieved point cloud learning by 2D convolutions on local tangent planes. KPConv networks [98] designed a point-based convolution kernel for point cloud learning. SparseConvNet [32] and MinkowskiCNN [16] have shown that one can train very deep networks with well-implemented sparse 3D convolutions, for point cloud processing. With diverse applications and representations of 3D data, we envision more research on deep networks for point clouds in the near future.

11.2.2.3 Deep Learning on Graphs

Recently there has been a strong interest to study how to generalize CNNs to graphs, and these new types of CNNs are called "graph CNNs" Graph CNNs take a graph with vertex functions as input. Conventional image CNNs can be viewed as graph CNNs on 2D regular grids of pixels, with RGB color as vertex values. There have been some previous works studying graph CNN on more general graphs instead of 2D regular grids [12, 21, 24, 38], and [8, 9, 64] have a special focus on near-isometric 3D shape graphs like human bodies. To generalize image CNNs, these works usually try to tackle the following three challenges: defining translation structures on graphs to allow parameter sharing, designing compactly supported filters on graphs, and aggregating multi-scale information. Their constructions of deep neural networks

usually fall into two types: spatial construction and spectral construction. In Sect. 11.5 we will see two concrete examples from both of those types.

11.3 Deep Learning on Regularly Structured 3D Data

To deal with irregular 3D data such as point clouds or meshes, one popular approach is to first convert irregular data to a regularly structured form. The two most popular conversions are rendering/projection (to single-view or multi-view images) and voxelization (to volumetric grids). With regular structures, we can adopt the successful convolutional neural networks to learn deep representations of 3D data.

In multi-view images, a polygon mesh is rendered into 2D images from multiple viewpoints before any representation learning. This conversion leads to two benefits. First, rendering circumvents artifacts in the original geometric representations that are usually unfriendly to traditional geometric algorithms. Some examples of the artifacts include close but disconnected shape parts (which breaks the topology), empty interior structure, and non-manifold geometry. Second, as rendering produces images, so we can leverage both maturely designed convolutional neural network (CNN) architectures, as well as their pretrained weights from large-scale natural image datasets. Besides these two benefits, from a more intuitive perspective, the process of analyzing multi-view images to understand 3D is similar to the process humans perceive a 3D object—we understand the full shape of an object by observing it from multiple angles.

Another popular way to convert irregular 3D data to a regular form is to "voxelize" a 3D point cloud or a mesh to a volumetric grid. In its simplest form, the voxelization process produces an occupancy grid where each voxel (a 3D cube in space) is associated with a binary value, where one indicates the presence of a point or the voxel being crossed by a face in the mesh, while zero indicates an empty space without any point or mesh face crossing it. After this quantization step, the original irregular data is converted to a 3D array, which can be consumed by a 3D convolutional neural network (3D CNN). A 3D CNN (also referred to as a volumetric CNN) is very similar to an image CNN, except it uses 3D convolutions instead of 2D ones. Compared with multi-view images, 3D volumetric inputs preserve original 3D geometry (in a quantized version) and are able to capture interior structures of objects. In certain applications of 3D data such as analysis of CT scans, 3D CNNs are the de facto approach since the input is naturally in volumetric forms.

In the following part of this section, we first introduce a deep neural network that consumes multi-view images called Multi-view CNN (MVCNN) in Sect. 11.3.1. Then in Sect. 11.3.2, we go into details of volumetric CNNs/3D CNNs and describe a simple but effective volumetric CNN called VoxNet. Lastly in Sect. 11.3.3, we compare MVCNN and volumetric CNNs on the 3D object classification task, where we also discuss the gap between their performance. In the same section, we further provide more experiments to understand the gap, as well as introduce two new and more powerful volumetric CNNs trying to fill the gap.

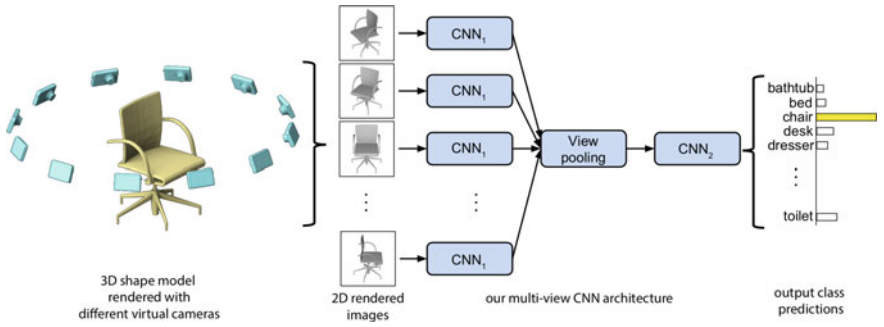


Fig. 11.2 Multi-view CNN for 3D shape recognition. At test time a 3D shape is rendered from 12 different views and is passed through CNN_1 to extract view based features. These are then pooled across views and passed through CNN_2 to obtain a compact shape descriptor. Figure is from [92]

11.3.1 Multi-view CNN

The work from Su et al. [92] proposed a simple but effective way to learn 3D shape descriptors for classification and retrieval with a multi-view representation of the 3D data. Because of the input data format and the use of CNNs to consume multi-view images, the proposed network is called Multi-view CNN (MVCNN). The system pipeline and the network architecture is shown in Fig. 11.2. In the MVCNN shape recognition pipeline, we first generate multiple views of a 3D shape by a rendering engine. Then MVCNN learns an aggregated representation combining features from multiple views, as a compact descriptor for the 3D shape. The learning process is supervised by a 3D shape classification task. The detailed process is explained in the following two subsections.

Note that compared with the proposed approach in MVCNN, there are several more naive approaches. For example, a simple way to use multiple views of a 3D shape for its classification is to generate 2D image descriptors from each view independently and then classify the shape based on these independent image descriptors, for example, with a voting scheme. Alternatively, if the input 3D shapes are aligned, we can render the 3D shape from a fixed set of viewpoints, extract 2D descriptors from each of the views and then concatenate the 2D descriptors in the same order as that of the views. We can use the concatenated vector as the shape descriptor and train a model to classify the shape based on it. However, aligning 3D shapes to a canonical orientation is hard by itself (or requires extra annotations). On the other hand, this approach cannot adapt to the varying number of views in case of partial 3D input. Compared to these two approaches, MVCNN is able to *learn* to aggregate views and is able to take the varying number of views. Details of the MVCNN pipeline is described below.

11.3.1.1 Multi-view Image Generation

To generate a multi-view representation of a 3D shape, we have to consider what renderer to use and the choice of rendering parameters including camera position and scene lighting.

As to the renderer, the original MVCNN paper chooses to use the Phong reflection model [73] and renders mesh polygons with a perspective projection. The pixel color is determined by interpolating the reflected intensity of the polygon vertices. Although simple, these rendered views with Phong reflection are very informative to summarize the geometry of the shape. For textured meshes, we can also use a more sophisticated renderer such as a ray traced renderer to respect more details in color and material properties. One can even generate beyond the common gray or RGB images, such as depth images, normal maps, etc., to capture more geometric properties of the shapes. For simplicity, in the following discussion and experiments we assume the shapes are rendered into gray images.

In the rendering scene the 3D shape is scaled into the same size and fits into a viewing volume at the center of the scene. In rendering engines with required scene lighting, we can uniformly place several point light sources around the 3D shape [75]. As long as renderings of all 3D shapes share the same lighting setup, the learned global 3D descriptors will be comparable. In another case, if we also want to test the deep network on views from natural images or under unknown lighting conditions, using a fixed lighting setup will cause issues as the model would overfit to the rendering process thus will not generalize to new images with different lighting. One way to resolve the issue is to randomly change the scene lighting parameters such as the number, energy, and color spectrum of the light sources, a process known as domain randomization [99].

Another key factor of generating multi-view representation is to choose the camera poses (viewpoints) for rendering. There are two common camera setups depending on our prior knowledge of the input 3D shape. In the first camera setup, we assume that the input shapes are upright oriented along a consistent axis (e.g., the z-axis). Most man-made CAD models in shape repositories such as the 3D Warehouse and datasets like ModelNet40 and ShapeNet satisfy this assumption. Based on this assumption, we render K views by placing K virtual cameras evenly spread around the 3D shape. Empirically, MVCNN takes $K = 12$ views such that every view is 30° from each other, and they are elevated 30° from the ground plane pointing to the center of the object (see Fig. 11.2). The center of the object is the weighted average of the mesh face vertices where the weight is the face area. For more complicated shapes (like mechanical components), we can also place another set of cameras elevated $-30°$ to capture its bottom geometry. In the second camera setup, we do not assume an upright orientation of the object, i.e., we have no prior knowledge of the shape's orientation. In this case we have to place more cameras than those in the first case in order not to miss representative views of the object. As in [92], we can place 20 virtual cameras at the 20 vertices of an icosahedron enclosing the shape, with all of them pointing to the center of the shape. Then we generate 4 rendered views from

each camera using 0, 90, 180, 270° rotations along the optical axis (in-tilt rotations), yielding a total of 80 views.

As noted in [92], using different shading coefficients or illumination models did not have much effect on the learned 3D descriptors due to the invariance of the learned filters to illumination changes, as also observed in image-based CNNs. Also the currently proposed camera setups are already able to achieve successful results—adding more or different viewpoints is trivial but leads to marginal performance improvement. Finally, by using the simple Phong reflection model, rendering each mesh from all the viewpoints takes no more than ten milliseconds on modern graphics hardware [92].

11.3.1.2 Multi-view CNN Architecture

Given multi-view images of an object, a simple baseline method to classify the shape is to extract features from each individual image and then aggregate their features (similar to light-field descriptor) for a further linear classifier. Or one can predict classification scores (with a linear classifier, a SVM) separately for each image and then "bagging" the results by taking the average of scores from all images. While these two baselines work well, the information from multi-view images is not fully exploited. Multi-view CNN further exploits the input by learning to aggregate views. The architecture (as illustrated in Fig. 11.2) mainly has three parts: the image feature extraction (with CNN_1), view pooling, and 3D descriptor learning (with CNN_2).

As in Fig. 11.2, first, each image in the multi-view representation is processed by the first part of the network (CNN_1) separately. The same CNN_1 is shared across all views. Then image descriptors extracted by CNN_1 are aggregated through a view-pooling layer, which is simply an element-wise maximum operation. An alternative view-pooling is an element-wise mean operation, but its empirical results are not as effective. Then the pooled image descriptor is fed to another CNN_2 for further processing to form a final 3D shape descriptor. The 3D shape descriptor can readily be used for many different tasks, such as for shape classification (with a linear softmax classifier), or shape retrieval. As the entire architecture is a directed acyclic graph, the entire model can be trained end-to-end using stochastic gradient descent with back-propagation. The supervision signal is from an annotated dataset (ModelNet [110]) with shape category labels.

In implementation, it is possible to select different architectures for the subnetworks CNN_1 and CNN_2. However, in order to leverage the mature architecture design, it is recommended to split existing image CNN architectures to form CNN_1 and CNN_2, in which case we can also take advantage of the model's pretrained weights from much larger image datasets such as ImageNet [23]. For example, in the original work of MVCNN [92], the authors adopted VGG [88] architecture using its conv1 to conv5 layers to construct CNN_1 and the rest of the layers to construct CNN_2, with the fc7 output (after ReLU nonlinearity) as the final 3D shape descriptor. The VGG network is pretrained on ImageNet images from 1k categories and then

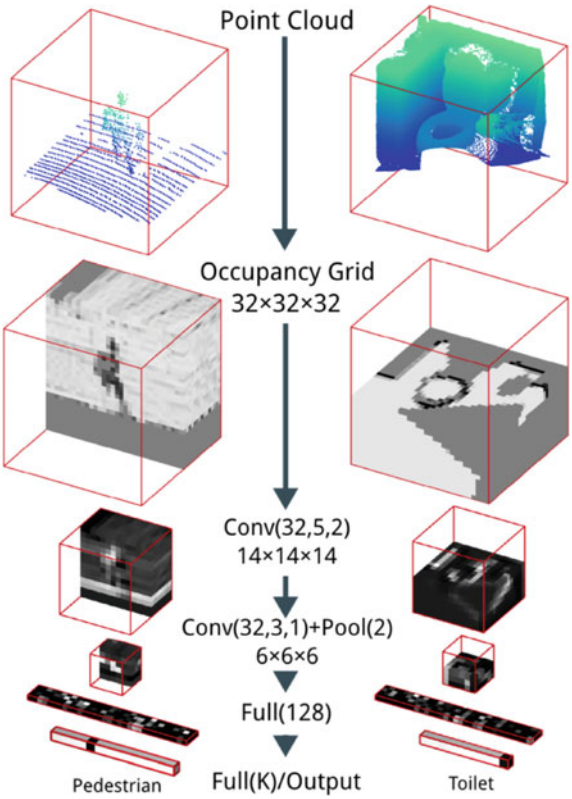


Fig. 11.3 VoxNet architectures. Conv(f, d, s) indicates f filters of size d and at strides s, Pool(m) indicates pooling with area m, and Full(n) indicates fully connected layer with n outputs. Inputs, example feature maps, and predicted outputs are shown. The point cloud on the left is from LiDAR and the one on the right is from a RGB-D scan. Cross sections are used for visualization purposes. Figure is from [65]

fine-tuned on all 2D views of the 3D shapes in the training set, with a classification task.

In Sect. 11.3.3, we will present a quantitative evaluation of the MVCNN architecture on a shape classification benchmark and compare it with the volumetric CNNs introduced in the next section.

11.3.2 Volumetric CNN

In this section, we present a simple, effective, and representative volumetric CNN/3D CNN architecture for object classification. The architecture called *VoxNet* [65] takes a volumetric occupancy grid, extracts hierarchical features from the 3D grid input, and finally gets a global descriptor of the 3D input for shape classification. In the following text, we first discuss how to prepare volumetric data from point clouds or polygon meshes, and then show the specific architecture design of VoxNet.

11.3.2.1 Voxelization

Voxelization is a discretization process that converts geometric data in a continuous domain (such as point clouds and meshes) into a discrete grid (a volumetric array). Depending on the input source, there are multiple ways to voxelize the irregular 3D data into volumetric data.

Given a point cloud, the simplest way of voxelization is to compute a binary occupancy grid based on whether there is any point in a voxel, which is called a hit grid in [65]. If we also know the pose of the sensor (such as a LiDAR or a depth camera) that acquires the point cloud, we may also encode the voxel as free (observed as empty), occupied, and unknown (occluded) through 3D ray tracing. There is also a more advanced way to compute an occupancy grid in a probabilistic way or compute a density grid with continuous values in each voxel (more details can be found in [65]).

We can also voxelize a polygon mesh (typically with triangles) into a volumetric occupancy grid. The first step is to subdivide the faces in the mesh until the longest edge of a face is small enough (e.g., half the length of a voxel size), and then set the voxels beneath the face vertices to one to create a "filled" voxelization (assuming space inside the object is occupied), or set voxels with vertices in them as occupied to create a "hollow" voxelization (assuming space inside the object is empty). Although the binary grid is simple and effective in many cases, we can encode more geometric information with a continuous value grid to enrich each voxel's information. The most popular encoding is a distance field and its variants, such as a truncated signed distance field (TSDF). In TSDF volumetric representation, each voxel stores a truncated distance from the voxel center to its nearest mesh surface, with a positive number if the voxel is outside or a negative number if the voxel is inside the surface. If there is no orientation in the faces, we can also build an unsigned distance field (DF) [20].

Without special notice, in the following sections, we assume we have a binary occupancy grid as input to all the volumetric CNNs.

11.3.2.2 VoxNet Architecture

The basic building blocks for a volumetric CNN include the input layer (preprocessing layer), 3D convolutional layers, pooling layers (average pooling or max pooling), fully connected layers, and up-convolution layers (omitted as up-convolutions are just strided convolutions). More detailed definitions are as follows:

The input layer takes a fixed size grid of $I \times J \times K$ voxels. Due to the computation and space complexity the resolution of the input grid is usually not high. For example, in [65], $I = J = K = 32$. The occupancy value is shifted by 0.5 and scaled by 2 so that they are in the range of $(-1, 1)$ (for binary occupancy grid the values are either -1 or 1). While for simplicity we only consider scalar valued inputs, we can easily add more feature channels for each voxel such as the LiDAR intensity values or the RGB color from an optical camera.

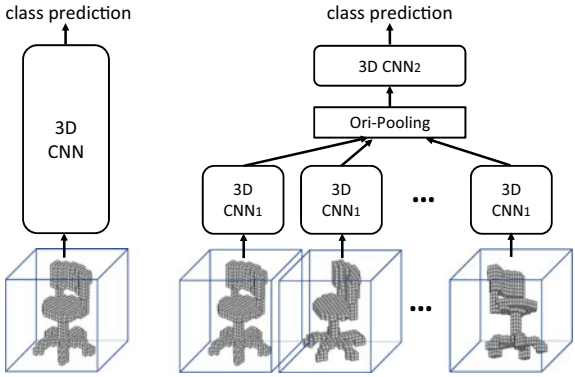


Fig. 11.4 Left: volumetric CNN (single orientation input). Right: multi-orientation volumetric CNN (MO-VCNN), which takes in various orientations of the 3D input, extracts features from shared CNN_1 and then pass pooled feature through another network CNN_2 to make a prediction

A 3D convolutional layer takes four-dimensional input volumes (height H, width W, length L, channel C), among them the first three dimensions are spatial and the last dimension contains the feature vectors for the voxels. In a batch mode (usually in GPU processing), the input becomes five-dimensional, with a batch size dimension B as the first dimension ($B \times H \times W \times L \times C$). A 3D convolution kernel (or filter) is in the size of $h \times w \times l \times C$, where h, w, l define the kernel size. Usually we use a square kernel with the same sizes on each side ($d \times d \times d \times C$). If we define F kernel functions for a given convolution layer, the total number of parameters (without counting the bias term) is $F \times d \times d \times d \times C$. The 3D convolution kernel will convolve with local volumetric regions by shifting its position in *3D space*, with stride and padding choices similar to those defined for 2D convolutions. In the VoxNet architecture, we use square kernels noted as Conv(f, d, s) where f is the number of kernels, d is its spatial dimension, and s is the stride size in shifting. The output of the convolution is passed through a leaky rectified nonlinearity layer (leaky ReLU) [62].

A 3D max-pooling layer Pool(m) downsamples the input volume by a factor of m along the three spatial dimensions. It replaces each $m \times m \times m$ nonoverlapping block of voxels with their maximum. There are also more parameter choices such as uneven spatial dimensions and strides, similar to 2D pooling layers.

A fully connected layer takes vector inputs and maps the vector to an output vector with a weight matrix multiplication, followed by a nonlinearity layer such as ReLU. In VoxNet, we use Full(n) to denote a fully connected layer with n outputs. In the last layer, the number of output matches the number of object categories and a softmax nonlinearity is used to provide a probabilistic score.

Based on the definitions of layers above, the VoxNet architecture is simply a composition of those layers with Conv$(32, 5, 2) \rightarrow$ Conv$(32, 3, 1) \rightarrow$ Pool$(2) \rightarrow$ Full$(128) \rightarrow$ Full(K) (Fig. 11.3) with K as the number of classes. The network is simple, and therefore, fast and portable, with just 921,736 parameters most of which are from the first fully connected layer (around 96% of the total parameters).

11.3.2.3 Tips to Improve Volumetric CNN Performance

There are several strategies to improve the performance of volumetric CNNs through data augmentation in training (to reduce overfitting), voting from multiple inputs, as well as learning to aggregate multi-orientation inputs and learning to align inputs.

Data Augmentation

Similar to learning on large image datasets, we can augment the training data with random rotation, shifting, and scaling, which helps reduce model overfitting in training. For example, we can create n copies of each input instance by rotating them with $360/n^\circ$ along the upright axis. We can also shift the input in the three spatial dimensions with a random small distance or scale the input by a random factor in the range of $[1-\delta, 1+\delta]$ where δ is a small number, for example, 0.1.

Voting and Learning to Vote

At testing time, we can pool the activations of the output layer over all n rotated versions of the input by taking the average of their classification scores. We can even take a similar approach to Multi-view CNN, to learn to aggregate features from observations from different orientations of the object, which can be called multi-orientation pooling (Fig. 11.4).

Learning to Align

An alternative approach of learning to vote is learning to align objects in different orientations to a "canonical" orientation before it is processed by the volumetric CNN. This can be achieved through a 3D spatial transformer network [75], which is a subnetwork (a volumetric CNN by itself but with fewer parameters) that takes volumetric grid input and output a transformation matrix (e.g., a rotation matrix) that is applied on the input. Then a following volumetric CNN consumes the rotated version of the input. The transformer network is jointly optimized with the following network and is shown to be able to learn to canonicalize object poses. With available annotations on object poses, we can even explicitly supervise the transformer network to regress to the ground truth pose, a relative pose to a predefined canonical pose (e.g., a chair with its back facing the front).

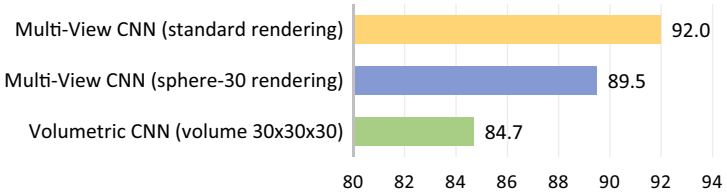


Fig. 11.5 Classification accuracy. Yellow and blue bars: performance drop of multi-view CNN due to the discretization of CAD models in rendering. Blue and green bars: volumetric CNN is significantly worse than multi-view CNN, even though their inputs have similar amounts of information. This indicates that the network of the volumetric CNN is weaker than that of the multi-view CNN

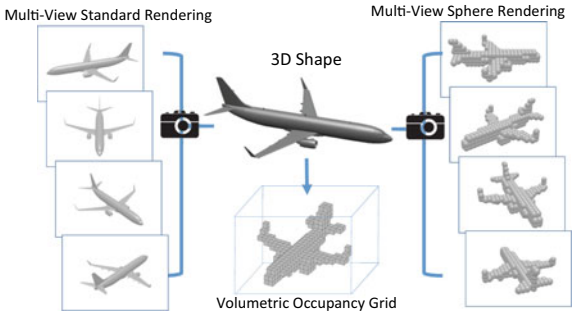


Fig. 11.6 3D shape representations

11.3.3 3D Volumetric CNN Versus Multi-view CNN

In the above two sections, we present two types of deep architecture on 3D shapes, volumetric and multi-view. In this section we make a shoulder to shoulder comparison of their performance, explain their performance gap, and briefly present two new volumetric deep networks with very different design intuitions to fill the gap.

The volumetric representation encodes a 3D shape as a 3D tensor of binary or real values. The multi-view representation encodes a 3D shape as a collection of renderings from multiple viewpoints. Stored as tensors, both representations can easily be used to train convolutional neural networks, i.e., volumetric CNNs and multi-view CNNs.

Intuitively, a volumetric representation should encode as much information, if not more, than its multi-view counterpart. However, experiments indicate that multi-view CNNs produce superior performance in object classification. Figure 11.5 reports the classification accuracy[1] on the ModelNet40 test set by two representative

[1] Average instance accuracy: the ratio between total number of correctly classified shapes and total number of tested shapes.

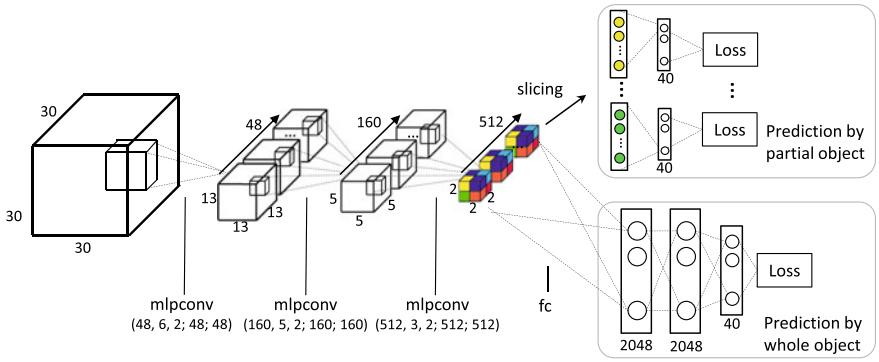


Fig. 11.7 Auxiliary training by subvolume supervision (Sect. 11.3.4.2). The main innovation is that we add auxiliary tasks to predict class labels that focus on part of an object, intended to drive the CNN to more heavily exploit local discriminative features. An mlpconv layer is a composition of three conv layers interleaved by ReLU layers. The five numbers under mlpconv are the number of channels, kernel size and stride of the first conv layer, and the number of channels of the second and third conv layers, respectively. The kernel size and stride of the second and third conv layers are 1. For example, mlpconv(48, 6, 2; 48; 48) is a composition of conv(48, 6, 2), ReLU, conv(48, 1, 1), ReLU, conv(48, 1, 1), and ReLU layers. Note that we add dropout layers with rate = 0.5 after fully connected layers

volumetric/multi-view architectures.[2] A volumetric CNN based on voxel occupancy (green) is 7.3% worse than a multi-view CNN (yellow).

It is interesting to ask why there is this performance gap. The gap seems to be caused by two factors: input resolution and network architecture differences. The multi-view CNN down-samples each rendered view to 227×227 pixels (Multi-view Standard Rendering in Fig. 11.6); to maintain a similar computational cost, the volumetric CNN uses a $30 \times 30 \times 30$ occupancy grid (Volumetric Occupancy Grid in Fig. 11.6).[3] As shown in Fig. 11.6, the input to the multi-view CNN captures more detail.

However, the difference in input resolution is not the primary reason for this performance gap, as evidenced by further experiments. If we compare the two networks by providing them with data containing a similar level of detail. To this end, we feed the multi-view CNN with renderings of the $30 \times 30 \times 30$ occupancy grid using *sphere rendering*,[4] i.e., for each occupied voxel, a ball is placed at its center, with radius equal to the edge length of a voxel. (Multi-View Sphere Rendering in Fig. 11.6.) We train the multi-view CNN from scratch using these sphere renderings. The accuracy of this multi-view CNN is reported in blue.

[2] We train models by replicating the architecture of [110] for volumetric CNNs and [92] for multi-view CNNs. All networks were trained in an end-to-end fashion. All methods were trained/tested on the same split for a fair comparison.

[3] Note that $30 \times 30 \times 30 \approx 227 \times 227 (\times 0.5)$.

[4] It is computationally prohibitive to match the volumetric CNN resolution to multi-view CNN, which would be $227 \times 227 \times 227$.

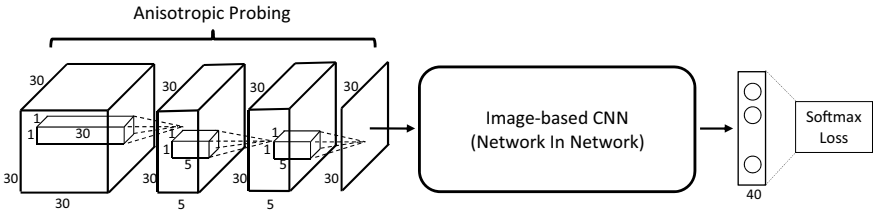


Fig. 11.8 CNN with anisotropic probing kernels. We use an elongated kernel to convolve the 3D cube and aggregate information to a 2D plane. Then we use a 2D NIN (NIN-CIFAR10 [59]) to classify the 2D projection of the original 3D shape. Figure is from [75]

Additionally, low-frequency information in 3D seems to be quite discriminative for object classification—it is possible to achieve 89.5% accuracy (blue) at a resolution of only $30 \times 30 \times 30$. This discovery motivates the architectures of a multi-resolution version of MVCNN.

11.3.4 Two New Volumetric Convolutional Neural Networks

11.3.4.1 Overview

We introduce two network variations that significantly improve the VoxNet architectures introduced in Sect. 11.3.2 on 3D volumetric data. These architecture designs are originally from [75]. The first network is designed to mitigate overfitting by introducing auxiliary training tasks, which are themselves challenging. These auxiliary tasks encourage the network to predict object class labels from partial subvolumes. Therefore, no additional annotation efforts are needed. The second network is designed to mimic multi-view CNNs, as they are strong in 3D shape classification. Instead of using rendering routines from computer graphics, the network projects a 3D shape to 2D by convolving its 3D volume with an anisotropic probing kernel. This kernel is capable of encoding long-range interactions between points. An image CNN is then appended to classify the 2D projection. Note that the training of the projection module and the image classification module is end-to-end. This emulation of multi-view CNNs achieves similar performance to them, using only standard layers in the CNN.

In order to mitigate overfitting from too many parameters, we adopt the mlpconv layer from [59] as the basic building block in both network variations (after each $d \times d \times d$ convolution layer, there are a few extra $1 \times 1 \times 1$ convolution layers following it, to further process the features maps).

11.3.4.2 Network 1: Auxiliary Training by Subvolume Supervision

Since current 3D datasets such as ModelNet40 are still limited in scale, we often observe significant overfitting when we train volumetric CNNs. When a volumetric CNN overfits to the training data, it has no incentive to continue learning. Reference [75] thus introduced auxiliary tasks that are closely correlated with the main task but are difficult to overfit, so that learning continues even if the main task is overfitted.

These auxiliary training tasks also predict the same object labels, but the predictions are made solely on a local subvolume of the input. Without complete knowledge of the object, the auxiliary tasks are more challenging, and can thus better exploit the discriminative power of local regions. This design is different from the classic multi-task learning setting of heterogeneous auxiliary tasks, which inevitably requires collecting additional annotations (e.g., conducting both object classification and detection [30]).

This design is implemented through an architecture shown in Fig. 11.7. The first three layers are mlpconv (multi-layer perceptron convolution) layers, a 3D extension of the 2D mlpconv proposed by [59]. The input and output of the mlpconv layers are both 4D tensors. Compared with the standard combination of linear convolutional layers and max-pooling layers, mlpconv has a three-layer structure and is thus a universal function approximator if enough neurons are provided in its intermediate layers. Therefore, mlpconv is a powerful filter for feature extraction of local patches, enhancing approximation of more abstract representations. In addition, mlpconv has been validated to be more discriminative with fewer parameters than ordinary convolution with pooling [59].

At the fourth layer, the network branches into two. The lower branch takes the whole object as input for traditional classification. The upper branch is a novel branch for auxiliary tasks. It slices the $512 \times 2 \times 2 \times 2$ 4D tensor (2 grids along x, y, z axes, and 512 channels) into $2 \times 2 \times 2 = 8$ vectors of dimension 512. We set up a classification task for each vector. A fully connected layer and a softmax layer are then appended independently to each vector to construct classification losses. A simple calculation shows that the receptive field of each task is $22 \times 22 \times 22$, covering roughly $2/3$ of the entire volume.

11.3.4.3 Network 2: Anisotropic Probing

The success of multi-view CNNs is intriguing. Multi-view CNNs first project 3D objects to 2D and then make use of well-developed 2D image CNNs for classification. Inspired by its success, [75] designed a neural network that is also composed of the two stages. However, while multi-view CNNs use external rendering pipelines from computer graphics, the proposed network achieves the 3D-to-2D projection using differentiable layers,[5] in a manner similar to 'X-ray scanning'.

[5] A rising field related to this differentiable way of projection is called differentiable rendering [61] or neural rendering [49].

Key to this network is the use of an elongated anisotropic kernel which helps capture the global structure of the 3D volume. As illustrated in Fig. 11.8, the neural network has two modules: an anisotropic probing module and a network in network (NIN) module. The anisotropic probing module contains three convolutional layers of elongated kernels, each followed by a nonlinear ReLU layer. Note that both the input and output of each layer are 3D tensors.

In contrast to traditional isotropic kernels, an anisotropic probing module has the advantage of aggregating long-range interactions in the early feature learning stage with fewer parameters. As a comparison, with traditional neural networks constructed from isotropic kernels, introducing long-range interactions at an early stage can only be achieved through large kernels, which inevitably introduce many more parameters. After anisotropic probing, we use an adapted NIN network [59] to address the classification problem.

The anisotropic probing network is capable of capturing internal structures of objects through its X-ray like projection mechanism. This is an ability not offered by standard rendering. Combined with multi-orientation pooling (introduced below), it is possible for this probing mechanism to capture any 3D structure, due to its relationship with the Radon transform.

In addition, this architecture is scalable to higher resolutions, since all its layers can be viewed as 2D. While 3D convolution involves computation at locations of cubic resolution, we maintain quadratic computational cost.

11.3.5 Experimental Results

We show evaluation of volumetric CNNs and multi-view CNNs on the ModelNet40 dataset [110]. For convenience in the following discussions, we define *3D resolution* to be the discretization resolution of a 3D shape. That is, a $30 \times 30 \times 30$ volume has 3D resolution 30. The sphere rendering from this volume also has a 3D resolution 30, though it may have a higher 2D image resolution.

By default, we report classification accuracy on all models in the test set (average instance accuracy). For comparisons with other previous work we also report average class accuracy.

11.3.5.1 Classification Results

We compare the newly introduced methods with several strong baselines (previous state of the art) for shape classification on the ModelNet40 dataset. In the following, we discuss the results within volumetric CNN methods and within multi-view CNN methods.

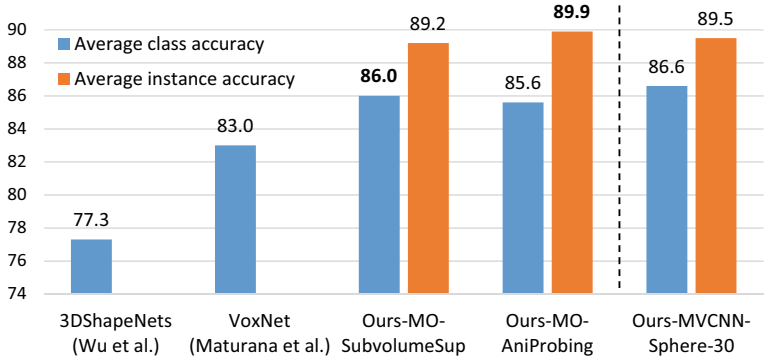


Fig. 11.9 Classification accuracy on ModelNet40 (resolution: 30). The newly introduced volumetric CNNs [75] have matched the performance of multi-view CNN at 3D resolution 30 (re-implementation of Su-MVCNN [92], rightmost group). Figure is from [75]

Volumetric CNNs

Figure 11.9 summarizes the performance of volumetric CNNs. MO-SubvolumeSup is the subvolume supervision network in Sect. 11.3.4.2 and MO-AniProbing is the anisotropic probing network in Sect. 11.3.4.3. Input data is augmented with random azimuth and elevation rotations. For clarity, we use MO- to denote that both networks are trained with an additional multi-orientation pooling step (20 orientations). For reference to multi-view CNN performance at the same 3D resolution, we also include MVCNN-Sphere-30, the result of the multi-view CNN with sphere rendering at 3D resolution 30.

As can be seen, both of these two new volumetric CNNs significantly outperform previous volumetric CNNs. Moreover, they both match the performance of multi-view CNN under the same 3D resolution. That is, *the gap between volumetric CNNs and multi-view CNNs is closed* under 3D resolution 30 on the ModelNet40 dataset, an issue that motivates us.

Multi-view CNNs

Figure 11.10 summarizes the performance of multi-view CNNs. MVCNN-MultiRes is the result of training an SVM over the concatenation of fc7 features from MVCNN-Sphere-30, 60, and MVCNN. HoGPyramid-LFD is the result by training an SVM over a concatenation of HoG features at three 2D resolutions (227×227, 114×114, and 57×57). Here LFD (lightfield descriptor) simply refers to extracting features from renderings. MVCNN-MultiRes achieves state-of-the-art accuracy.

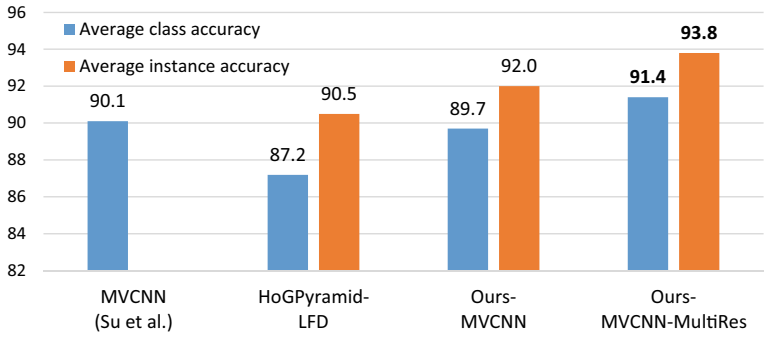


Fig. 11.10 Classification accuracy on ModelNet40 (multi-view representation). The 3D multi-resolution version is the strongest. It is worth noting that the simple baseline HoGPyramid-LFD performs quite well. Figure is from [75]

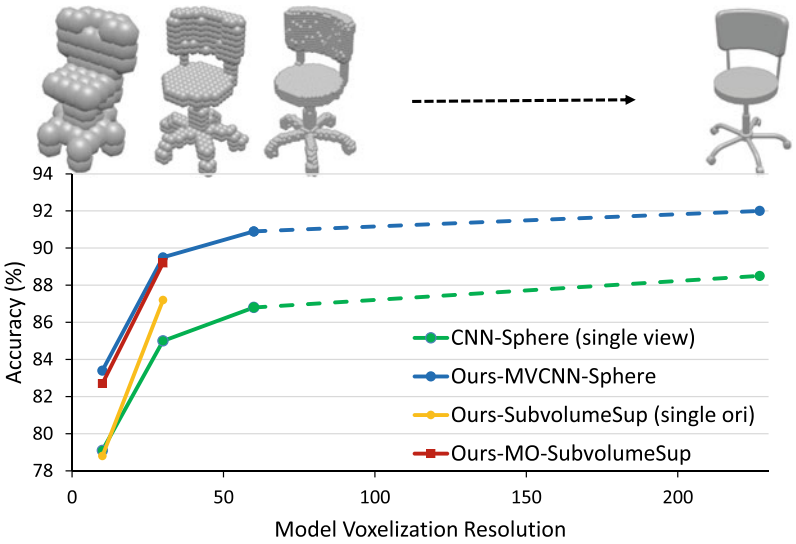


Fig. 11.11 Top: sphere rendering at 3D resolution 10, 30, 60, and standard rendering. Bottom: performance of image-based CNN and volumetric CNN with increasing 3D resolution. The two rightmost points are trained/tested from standard rendering. Figure is from [75]

11.3.5.2 Effect of 3D Resolution over Performance

Section 11.3.5.1 shows that the new volumetric CNN and multi-view CNN perform comparably at 3D resolution 30. Here we study the effect of 3D resolution for both types of networks.

Figure 11.11 shows the performance of volumetric CNN and multi-view CNN at different 3D resolutions (defined at the beginning of Sect. 11.3.5). Due to computational cost, we only test volumetric CNNs at 3D resolutions 10 and 30. The

Table 11.1 Comparison of volumetric CNN architectures. Numbers reported are classification accuracy on ModelNet40. Results from E2E-[110] (end-to-end learning version). All experiments are using the same set of azimuth and elevation augmented data

Network	Single-orientation	Multi-orientation
E2E-[110]	83.0	87.8
VoxNet [65]	83.8	85.9
3D-NIN	86.1	88.5
SubvolumeSup	**87.2**	89.2
AniProbing	84.4	**89.9**

observations are: first, the performance of volumetric CNNs and multi-view CNNs is on par at tested 3D resolutions; second, the performance of multi-view CNNs increases as the 3D resolution goes up. To further improve the performance of volumetric CNNs, this experiment suggests that it is worth exploring how to scale up volumetric CNNs to take input with higher 3D resolutions.

Comparison of Volumetric CNN Architectures

The architectures in comparison include VoxNet [65], E2E-[110] (the end-to-end learning variation of [110] implemented in Caffe [46]), 3D-NIN (a 3D variation of Network in Network [59] designed for 3D as in Fig. 11.7 without the "Prediction by partial object" branch), SubvolumeSup (Sect. 11.3.4.2) and AniProbing (Sect. 11.3.4.3).

From Table 11.1, first, the two volumetric CNNs we propose, SubvolumeSup and AniProbing networks, both show superior performance, indicating the effectiveness of the new architecture design; second, multi-orientation pooling increases performance for all network variations. This is especially significant for the anisotropic probing network, since each orientation usually only carries partial information of the object.

Comparison of Multi-view Methods

We compare different methods that are based on multi-view representations in Table 11.2. Methods in the second group are trained on the full ModelNet40 train set. Methods in the first group, SPH, LFD, FV, and Su-MVCNN, are trained on a subset of ModelNet40 containing 3,183 training samples. They are provided for reference. Also, note that the MVCNNs in the second group are re-implementations of MVCNN [92] in Caffe with AlexNet instead of VGG. We observe that MVCNNs are superior to methods by SVMs on handcrafted features.

Table 11.2 Comparison of multi-view based methods. Numbers reported are classification accuracy (class average and instance average) on ModelNet40. Note here the MVCNN is a replicated version of [92] trained on a larger training set

Method	#Views	Accuracy (class)	Accuracy (instance)
SPH (reported by [110])	–	68.2	–
LFD (reported by [110])	–	75.5	–
FV (reported by [92])	12	84.8	–
Su-MVCNN [92]	80	90.1	–
PyramidHoG-LFD	20	87.2	90.5
MVCNN	20	89.7	92.0
MVCNN-MultiRes	20	**91.4**	**93.8**

11.3.6 Further Discussion

In the text above we have introduced two types of deep neural networks to analyze 3D data: multi-view CNNs and volumetric CNNs (3D CNNs) and specifically described one multi-view CNN architecture: MVCNN from [92] and three volumetric CNNs: VoxNet from [65] and two stronger performing models from [75].

While we focused mainly on the semantic classification task, these two types of deep networks can be extended to many other applications such as shape part segmentation [48] and semantic scene segmentation [18] by aggregating multi-view image segmentation results, and 3D object detection [89] or 3D scan completion [20, 90] with volumetric CNNs.

These two types of networks have their own pros and cons as well. Multi-view CNNs are often good at capture fine-grained geometric structures (such as keys in a keyboard or thin bars on a chair back) thanks to the high-resolution image input and are very discriminative due to the well engineered and pretrained CNNs available—that's why MVCNNs are the leading methods in shape retrieval challenges such as the SHREC'16 [84]. However, multi-view CNNs have more strict requirements on the input format than volumetric CNNs. When the 3D input is not in the form of meshes or high density point clouds, or when the input is partial scans, it is not obvious how to render or project the input into multi-view images. For 3D input at the scene level, how to balance the quality, quantity, and efficiency of view-rendering is also a challenge. On the other hand, volumetric CNNs are more flexible with the input types (meshes, point clouds; partial or complete; single objects or scenes), as essentially it is just a quantization step. However, the main challenge with volumetric CNNs is on computation cost with high-resolution input as we discussed in the experiment subsection above. The network architecture is also coupled with the input resolution, i.e., adjusting the input resolution changes the physical receptive field sizes of the network layers. Recently there are a couple of new projects [16, 32] that proposed

to use sparse 3D convolutions to achieve high efficiency while keeping high spatial resolution, which got promising results especially on the large-scale semantic scene parsing task [19].

A shared concern with both types of these networks (multi-view and volumetric CNNs) is that they all require a preprocessing step of the raw 3D point clouds or meshes before representation learning. These preprocessing steps on one hand could cause loss of 3D information and on the other hand lead to more hyper-parameter choices (e.g., viewpoints, rendering parameters or voxelization resolution) that are not obvious to determine in many scenarios. In the next two sections (Sects. 11.4 and 11.5) we introduce two new types of deep neural networks that directly consumes point cloud or mesh data without converting them to regular structures.

11.4 Deep Learning on Point Clouds

In this section, we explore deep learning architectures capable of reasoning about point clouds, a popular and important type of 3D geometric data. Typical convolutional deep networks require regular input data formats, like those of image grids or 3D voxels, in order to perform weight sharing and other kernel optimizations. Since point clouds are not in a regular format, most researchers typically transform such data to regular 3D voxel grids or collections of images (e.g., views) before feeding them to a deep net architecture as we discussed in Sect. 11.3. This data representation transformation, however, renders the resulting data unnecessarily voluminous—while also introducing quantization artifacts that can obscure natural invariances of the data.

For this reason, in this chapter, we focus on directly processing point clouds without converting them to other formats—and mainly introduce one representative deep network named *PointNet* [76]. The PointNet architecture directly takes point clouds as input and outputs either class labels for the entire input or per-point segment/part labels for each point of the input (Fig. 11.12).

Although point clouds are simple in representation, we still face two challenges in the deep architecture design. First, the model needs to respect the fact that a point cloud is just a set of points, and therefore, invariant to permutations of its members. Second, invariances to rigid transformations also need to be considered. To address the challenges, PointNet is constructed as a symmetric function composed of neural networks, which guarantees its invariance to input point orders. Furthermore, the input format is easy to apply rigid or affine transformations to, as each point transforms independently. Thus we can add a data-dependent spatial transformer network that attempts to canonicalize the data before the PointNet processes them, so as to further improve the results.

We provide both theoretical analysis and an experimental evaluation of the approach. We show that PointNet can approximate any set function that is continuous. More interestingly, it turns out that the network learns to summarize an input point cloud by a sparse set of key points, which roughly corresponds to the skele-

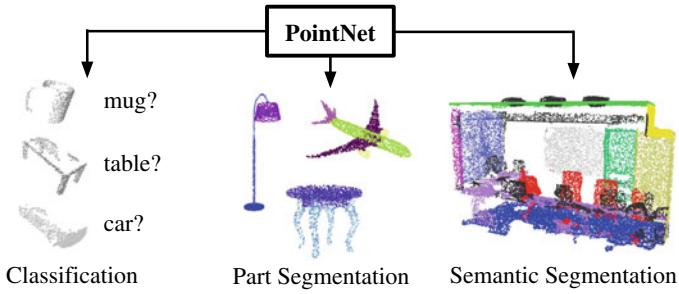


Fig. 11.12 Applications of PointNet. We propose a novel deep net architecture that consumes raw point clouds (sets of points) without voxelization or rendering. It is a unified architecture that learns both global and local point features, providing a simple, efficient, and effective approach for a number of 3D recognition tasks. Figure is from [76]

ton of objects according to visualization. The theoretical analysis and visualizations provide an understanding of why PointNet is highly robust to small perturbation of input points as well as to corruption through point insertion (outliers) or deletion (missing data).

We show the experimental results of PointNet on a number of benchmark datasets ranging from shape classification, part segmentation to scene segmentation, and compare it with deep learning methods with multi-view and volumetric representations. Under a unified architecture, the PointNet is much faster in speed, but it also exhibits strong performance on par or even better than the prior art.

In the rest of the section, Sect. 11.4.1 describes the problem formulation for deep learning on point clouds. Section 11.4.2 introduces the properties of point sets for which we refer to in the PointNet architecture design. Section 11.4.3 presents the detailed architecture of PointNet while Sect. 11.4.4 provides some theoretical analysis of the network. Lastly in Sect. 11.4.5 we show the experimental results of PointNet on shape recognition and semantic parsing benchmarks, and provide visualizations, as well as analysis experiments.

11.4.1 Problem Statement

We introduce a deep learning framework that directly consumes unordered point sets as inputs. A point cloud is represented as a set of 3D points $\{P_i \mid i = 1, \ldots, n\}$, where each point P_i is a vector of its (x, y, z) coordinate plus extra feature channels such as color, normal, etc. For simplicity and clarity, unless otherwise noted, we only use the (x, y, z) coordinate as point's channels.

For the object classification task, the input point cloud is either directly sampled from a shape or pre-segmented from a scene point cloud. The deep network outputs k scores for all the k candidate classes. For semantic segmentation, the input can be

a single object for part region segmentation, or a subvolume from a 3D scene for object region segmentation. The model will output $n \times m$ scores for each of the n points and each of the m semantic subcategories.

11.4.2 Properties of Point Sets in $\mathbb{R}^N$

In theory a fully connected neural network with one hidden layer is able to approximate any continuous function, however, injecting more structure in the network architecture can greatly reduce the number of necessary parameters and improve approximation accuracy. A well-known example is the design of convolutional neural networks for images. To find out a proper architecture structure, we need to first understand the properties of the data.

The input is a subset of points from an Euclidean space. It has three main properties:

- Unordered. Unlike pixel arrays in images or voxel arrays in volumetric grids, a point cloud is a set of points without a specific order. In other words, a network that consumes n 3D points needs to be invariant to $n!$ permutations of the input.
- Interaction among points. The points are from a space with a distance metric. This means that points are not isolated, and neighboring points form a meaningful subset. Therefore, the model needs to be able to capture local structures from nearby points, and the combinatorial interactions among local structures.
- Invariance under transformations. As a geometric object, the learned representation of the point set should be invariant to certain transformations. For example, rotating and translating points all together should not modify the global point cloud category nor the segmentation of the points.

11.4.3 PointNet Architecture

The full PointNet architecture is illustrated in Fig. 11.13, where the classification network and the segmentation network share a great portion of the common structure. Please read the caption of Fig. 11.13 for the pipeline.

The network has three key modules: the max-pooling layer as a symmetric function to aggregate information from all the points, a local and global information combination structure, and two joint alignment networks that align both input points and point features. We will discuss the reason behind these design choices in separate paragraphs below.

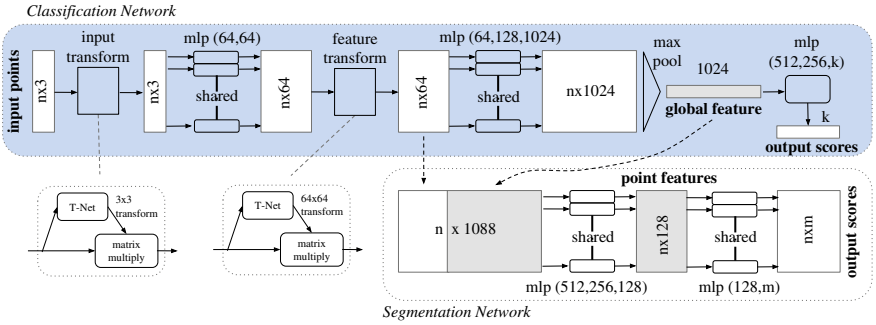


Fig. 11.13 PointNet architecture. The classification network takes n points as input, applies input and feature transformations, and then aggregates point features by max pooling. The output is classification scores for k classes. The segmentation network is an extension of the classification net. It concatenates global and local features and outputs per-point scores. "mlp" stands for multi-layer perceptron, numbers in bracket are layer sizes. Batchnorm is used for all layers with ReLU. Dropout layers are used for the last mlp in the classification net. Figure is from [76]

Symmetry Function for Unordered Input

In order to make a model invariant to input permutation, three strategies exist: (1) sort input into a canonical order; (2) treat the input as a sequence to train a RNN (recurrent neural network), but augment the training data by all kinds of permutations; (3) use a simple symmetric function to aggregate the information from each point. Here, a symmetric function takes n vectors as input and outputs a new vector that is invariant to the input order. For example, $+$ and $*$ operators are symmetric binary functions.

While sorting sounds like a simple solution, in high-dimensional space there, in fact, does not exist an ordering that is stable with respect to point perturbations in the general sense. This can be easily shown by contradiction. If such an ordering strategy exists, it defines a bijection map between a high-dimensional space and a $1D$ real line. It is not hard to see, to require an ordering to be stable with respect to point perturbations is equivalent to requiring that this map preserves spatial proximity as the dimension reduces, a task that cannot be achieved in the general case. Therefore, sorting does not fully resolve the ordering issue, and it is hard for a network to learn a consistent mapping from input to output as the ordering issue persists. As shown in experiments (Fig. 11.16), we find that a fully connected network applied on a sorted point set performs poorly, though slightly better than the one directly processing an unsorted input.

The idea to use an RNN considers the point set as a sequential signal and hopes that by training the RNN with randomly permuted sequences, the RNN will become invariant to input order. However, in "OrderMatters" [103] the authors have shown that order does matter and cannot be totally omitted. While an RNN has relatively good robustness to input ordering for sequences with small lengths (dozens), it's hard to scale up to thousands of input elements, which is the common size for point

sets. Empirically, we have also shown that a model based on RNNs does not perform as well as the proposed method (Fig. 11.16).

A more general idea is to approximate a general function defined on a point set by applying a symmetric function on transformed elements in the set

$$f(\{x_1, \ldots, x_n\}) \approx g(h(x_1), \ldots, h(x_n)), \tag{11.1}$$

where $f : 2^{\mathbb{R}^N} \to \mathbb{R}$, $h : \mathbb{R}^N \to \mathbb{R}^K$, $g : \mathbb{R}^K \times \cdots \times \mathbb{R}^K \to \mathbb{R}$ as a *symmetric function*. With this function composition, since g is symmetric and as h is shared across input elements $\{x_i\}$, the whole function f is guaranteed to be symmetric.

Empirically, the basic module is very simple: we approximate h by a multi-layer perceptron (MLP) network and g as a composition of (element-wise) max pooling ($\text{MAX} : \mathbb{R}^K \times \cdots \times \mathbb{R}^K \to \mathbb{R}^K$) and another multi-layer perceptron network $\gamma : \mathbb{R}^K \to \mathbb{R}$, therefore, $g = \gamma \circ \text{MAX}$.

The output dimension (K) of the function h is a key parameter as it specifies the latent space size for point aggregation. Empirically we take $K = 1024$. We also learn a number of f's, for example k of them as scores for k categories in the point cloud classification task. See Fig. 11.13 for detailed parameters of h and see visualizations (Fig. 11.20) in Sect. 11.4.5.3 for an intuitive explanation of what has been learned in the function h.

While the key module seems simple, it has interesting properties (see Sect. 11.4.5.3) and can achieve strong performance (see Sect. 11.4.5.1) in several different applications. Due to the simplicity of the module, we are also able to provide theoretical analysis as in Sect. 11.4.4.

Local and Global Information Aggregation

The output from the above max-pooling operator gives us a global signature of the input set. We can easily train a SVM or a multi-layer perceptron classifier (with γ and a linear layer) on this global feature for classification. However, point segmentation requires a combination of local and global knowledge (as a global descriptor cannot tell classes of individual points while a local descriptor has limited context). We can achieve this in a simple yet highly effective manner.

The solution can be seen in Fig. 11.13 (*Segmentation Network*). After computing the global point cloud feature after max pooling (1024-dim), we feed it back to per-point embeddings (64-dim) by concatenating the global feature with each of the points embeddings. Then we extract new per-point features based on the combined point descriptors—this time the per-point feature is aware of both the local and global information.

With this modification the network is able to predict per-point quantities that rely on both local geometry and global semantics. For example we can accurately predict per-point normals (see appendix in [76]), validating that the network is able to summarize information from the point's local neighborhood. In Sect. 11.4.5, we also

show that the model can achieve superior performance on shape part segmentation and scene segmentation compared with prior work.

Joint Alignment Network

The semantic labeling of a point cloud has to be invariant if the point cloud undergoes certain geometric transformations, such as rigid transformation. We, therefore, expect that the learned representation of a point set is invariant to these transformations.

A natural solution is to align all input set to a canonical space before feature extraction. Jaderberg et al. [45] introduced the idea of the spatial transformer to align 2D images through sampling and interpolation, achieved by a specifically tailored layer implemented on GPU.

The input form of point clouds allows us to achieve this goal in a much simpler way compared with [45]. We do not need to invent any new layers and no alias is introduced as in the image case. We predict an affine transformation matrix by a mini-network (T-net in Fig. 11.13) and directly apply this transformation to the coordinates of input points. The mini-network itself resembles the form of (11.1), composed of point embedding functions, max pooling, and fully connected layers. It takes a point cloud input and outputs a 3×3 transformation matrix (or a 64×64 matrix in the feature transformation case described below). More details about the T-net are described in the appendix in [76].

By training the T-net along with the rest of the network, the T-net learns to geometrically transform point clouds such that they are more aligned (typically to a few canonical poses within each category) so that the classification task afterward becomes easier.

This idea can be further extended to the alignment of feature space as well. We can insert another alignment network on point features and predict a feature transformation matrix to align features from different input point clouds. However, the transformation matrix in the feature space has a much higher dimension than the spatial transform matrix, which greatly increases the difficulty of optimization. We, therefore, add a regularization term to the softmax training loss. We constrain the feature transformation matrix to be close to an orthogonal matrix

$$L_{reg} = \|I - AA^T\|_F^2, \tag{11.2}$$

where A is the feature alignment matrix predicted by a mini-network. An orthogonal transformation will not lose information in the input, thus it is desired. We find that by adding the regularization term, the optimization becomes more stable and the model achieves better performance.

11.4.4 Theoretical Analysis

Universal Approximation

We first show the universal approximation ability of PointNet to continuous set functions. By the continuity of set functions, intuitively, a small perturbation to the input point set should not greatly change the function values, such as classification or segmentation scores.

Formally, let $\mathcal{X} = \{S : S \subseteq [0, 1] \text{ and } |S| = n\}$, $f : \mathcal{X} \to \mathbb{R}$ is a continuous set function on $\mathcal{X}$ w.r.t to Hausdorff distance $d_H(\cdot, \cdot)$, i.e., $\forall \epsilon > 0, \exists \delta > 0$, for any $S, S' \in \mathcal{X}$, if $d_H(S, S') < \delta$, then $|f(S) - f(S')| < \epsilon$. The theorem says that f can be arbitrarily approximated by the network given enough neurons at the max-pooling layer, i.e., K in (11.1) is sufficiently large.

Theorem 11.1 *Suppose $f : \mathcal{X} \to \mathbb{R}$ is a continuous set function w.r.t Hausdorff distance $d_H(\cdot, \cdot)$. $\forall \epsilon > 0$, $\exists$ a continuous function h and a symmetric function $g(x_1, \ldots, x_n) = \gamma \circ MAX$, such that for any $S \in \mathcal{X}$,*

$$\left| f(S) - \gamma \left(\underset{x_i \in S}{MAX} \{h(x_i)\} \right) \right| < \epsilon$$

where $x_1, \ldots, x_n$ is the full list of elements in S ordered arbitrarily, γ is a continuous function, and MAX is a vector max operator that takes n vectors as input and returns a new vector of the element-wise maximum.

The proof of this theorem can be found in the appendix of [76]. The key idea is that in the worst case the network can learn to convert a point cloud into a volumetric representation, by partitioning the space into equal-sized voxels. In practice, however, the network learns a much smarter strategy to probe the space, as we shall see in point function visualizations.

Bottleneck Dimension and Stability

Theoretically and experimentally we find that the expressiveness of the network is strongly affected by the dimension of the max-pooling layer, i.e., K in (11.1). Here we provide an analysis, which also reveals properties related to the stability of the model.

We define $\mathbf{u} = \underset{x_i \in S}{\text{MAX}}\{h(x_i)\}$ to be the subnetwork of f which maps a point set in $[0, 1]^m$ to a K-dimensional vector. The following theorem tells us that small corruptions or extra noise points in the input set are not likely to change the output of the network

Theorem 11.2 *Suppose $\mathbf{u} : \mathcal{X} \to \mathbb{R}^K$ such that $\mathbf{u} = \underset{x_i \in S}{MAX}\{h(x_i)\}$ and $f = \gamma \circ \mathbf{u}$. Then,*

(a) $\forall S, \exists\, \mathcal{C}_S, \mathcal{N}_S \subseteq \mathcal{X},\ f(T) = f(S)$ *if* $\mathcal{C}_S \subseteq T \subseteq \mathcal{N}_S$*;*
(b) $|\mathcal{C}_S| \leq K$

We explain the implications of the theorem. (a) says that $f(S)$ is unchanged up to the input corruption if all points in $\mathcal{C}_S$ are preserved; it is also unchanged with extra noise points up to $\mathcal{N}_S$. (b) says that $\mathcal{C}_S$ only contains a bounded number of points, determined by K in (11.1). In other words, $f(S)$ is in fact totally determined by a finite subset $\mathcal{C}_S \subseteq S$ of less or equal to K elements. We, therefore, call $\mathcal{C}_S$ the *critical point set* of S and K the *bottleneck dimension* of f. Section 11.4.5.3 describes more details on how to compute $\mathcal{N}_S$ and $\mathcal{C}_S$ empirically and also provides some visualizations of a few sample critical sets.

Combined with the continuity of h, this explains the robustness of the model with respect to point perturbation, corruption, and extra noise points. The robustness is gained in analogy to the sparsity principle in machine learning models. *Intuitively, the network learns to summarize a shape by a sparse set of key points.* In the following section we see that the key points form the skeleton of an object.

11.4.5 Experimental Results

Experiments are divided into four parts. First, we show that PointNet can be applied to multiple 3D recognition tasks (Sect. 11.4.5.1). Second, we provide experiments to validate the network design (Sect. 11.4.5.2). At last, we visualize what the network learns (Sect. 11.4.5.3) and analyze time and space complexity (Sect. 11.4.5.4).

11.4.5.1 Applications

In this section we show how PointNet can be trained to perform 3D object classification, object part segmentation and semantic scene segmentation. Even though the network takes input with an unstructured, unordered data representation (point sets), PointNet is able to achieve comparable or even better performance compared with strong prior methods on benchmarks for several tasks.

3D Object Classification

The network learns a global point cloud feature that can be used for object classification. We evaluate the model on the ModelNet40 [110] shape classification benchmark. There are 12,311 CAD models from 40 man-made object categories, split into 9,843 for training and 2,468 for testing. While previous methods focus on volumetric and multi-view image representations, PointNet was the first deep architecture to directly work on raw point cloud data.

Table 11.3 Classification results on ModelNet40

	Input	#Views	Accuracy avg. class	Accuracy overall
SPH [50]	Mesh	–	68.2	–
3DShapeNets [110]	Volume	1	77.3	84.7
VoxNet [65]	Volume	12	83.0	85.9
Subvolume [75]	Volume	20	86.0	**89.2**
LFD [110]	Image	10	75.5	–
MVCNN [92]	Image	80	**90.1**	–
Baseline	Point	–	72.6	77.4
PointNet	Point	1	86.2	**89.2**

We uniformly sample 1024 points on mesh faces according to face areas[6] and scale the point cloud of each shape into a unit sphere. During training, we augment the point cloud on-the-fly by randomly rotating the object along the up-axis and jitter the position of each point by a Gaussian noise with zero mean and 0.02 standard deviation.

In Table 11.3, we compare the PointNet model with previous works, as well as a baseline using MLP on traditional features extracted from point cloud (point density, D2, shape contour, etc.). The PointNet model achieved the best performance among methods based on 3D input (volumetric and point cloud). With only fully connected layers and max pooling, the network gains a strong lead in inference speed and can be easily parallelized in GPU implementation. There is still a small gap between the PointNet method and multi-view based method (MVCNN [92]), which we think is due to the loss of fine geometry details that can be captured by rendered images.

3D Object Part Segmentation

Part segmentation is a challenging fine-grained 3D recognition task. Given a 3D scan or a mesh model, the task is to assign a part category label (e.g., chair leg, cup handle) to each point or face.

We evaluate on the ShapeNet part data set from [114], which contains 16,881 shapes from 16 categories, annotated with 50 parts in total. Most object categories are labeled with two to five parts. Ground truth annotations are labeled on sampled points on the shapes.

The part segmentation is formulated as a per-point classification problem. The evaluation metric is mIoU on points. For each shape S of category C, to calculate the shape's mIoU: For each part type in category C, compute IoU between ground

[6] To sample one point, we first randomly select a face. A face f_i is selected with probability $p_i = \text{area}(f_i)/\sum_{j=1}^{N} \text{area}(f_j)$. Then we randomly sample a point from the face.

Table 11.4 Segmentation results on ShapeNet part dataset. Metric is mIoU (%) on points. We compare PointNet with two traditional methods [109, 114] and a 3D fully convolutional network baseline. PointNet method achieved the best mIoU compared with the baselines

	Mean	Aero	Bag	Cap	Car	Chair	Ear	Guitar	Knife
#Shapes		2690	76	55	898	3758	69	787	392
Wu [109]	–	63.2	–	–	–	73.5	–	–	–
Yi [114]	81.4	81.0	78.4	77.7	**75.7**	87.6	61.9	**92.0**	85.4
3D CNN	79.4	75.1	72.8	73.3	70.0	87.2	63.5	88.4	79.6
PointNet	**83.7**	**83.4**	**78.7**	**82.5**	74.9	**89.6**	**73.0**	91.5	**85.9**
		Lamp	Laptop	Motor	Mug	Pistol	Rocket	Skate	Table
#Shapes		1547	451	202	184	283	66	152	5271
Wu [109]		74.4	–	–	–	–	–	-	74.8
Yi [114]		**82.5**	**95.7**	**70.6**	91.9	**85.9**	53.1	69.8	75.3
3D CNN		74.4	93.9	58.7	91.8	76.4	51.2	65.3	77.1
PointNet		80.8	95.3	65.2	**93.0**	81.2	**57.9**	**72.8**	**80.6**

truth and predicted part labeling (among points). If the union of ground truth and prediction points is empty, then count part IoU as 1. Then we average IoUs for all part types in category C to get mIoU for that shape. To calculate mIoU for the category, we take an average of mIoUs for all shapes in that category.

We compare the segmentation version PointNet (a modified version [76] of Fig. 11.13, *Segmentation Network*) with two traditional methods [109, 114] that both take advantage of pointwise geometry features and correspondences between shapes, as well as a 3D CNN baseline [76].

In Table 11.4, we report per-category and mean IoU(%) scores. We observe a 2.3% mean IoU improvement and the network beats the baseline methods in most categories.

We also perform experiments on simulated Kinect scans to test the robustness of these methods. For every CAD model in the ShapeNet part data set, we use Blensor Kinect Simulator [35] to generate incomplete point clouds from six random viewpoints. We train PointNet on the complete shapes and partial scans with the same network architecture and training setting. Results show that the network lost only 5.3% mean IoU. In Fig. 11.14, we present qualitative results on both complete and partial data. One can see that though partial data is fairly challenging, the predictions from PointNet are still reasonable.

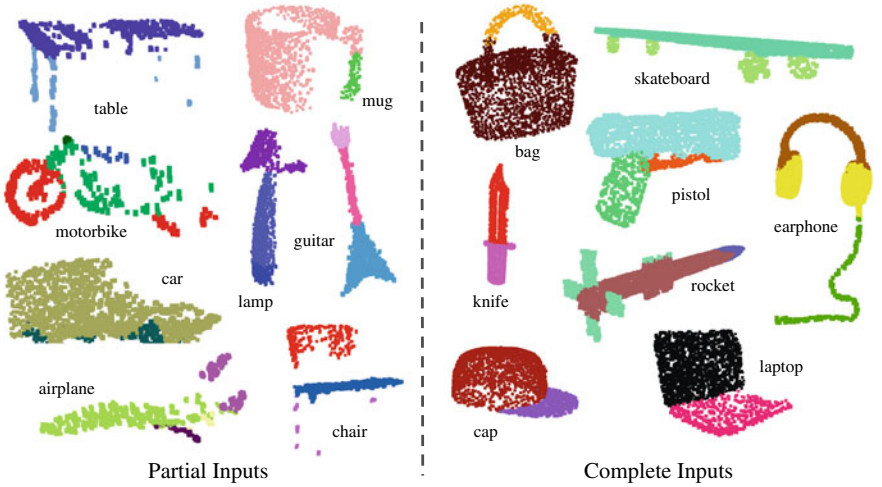


Fig. 11.14 Qualitative results for part segmentation. We visualize the CAD part segmentation results across all 16 object categories. We show both results for partial simulated Kinect scans (left block) and complete ShapeNet CAD models (right block). Figure is from [76]

Fig. 11.15 Qualitative results for semantic segmentation. Top row is an input point cloud with color. Bottom row is output semantic segmentation result (on points) displayed in the same camera viewpoint as input

Semantic Segmentation in Scenes

The segmentation PointNet on part segmentation can be easily extended to semantic scene segmentation, where point labels become semantic object classes instead of object part labels.

We show the experimental results on the Stanford 3D Indoor Spaces (S3DIS) dataset [3], which contains 3D scans from Matterport scanners in 6 areas (building floors) including 271 rooms annotated with 13 semantic categories.

To prepare training data, we first split points by room, and then split each room into blocks with 1 m by 1 m areas. We train the segmentation version of PointNet to predict a per-point class in each block. More details of the experimental setup can be found in [76].

Figure 11.15 shows the segmentation results from PointNet. We see the network is able to output smooth predictions and is robust to missing points and occlusions.

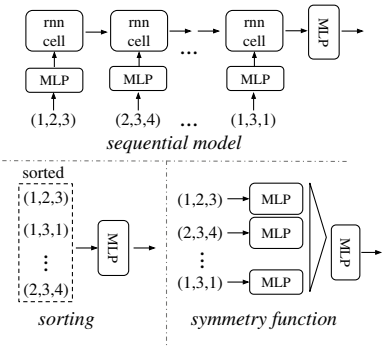
11.4.5.2 Architecture Design Analysis

In this section we validate the design choices of PointNet by control experiments. We also show the effects of the network's hyperparameters.

Comparison with Alternative Order-Invariant Methods

As mentioned in Sect. 11.4.3, there are at least three options for consuming unordered set inputs. We use the ModelNet40 shape classification problem as a test bed for comparisons of those options, the following two control experiment will also use this task.

The baselines (illustrated in Fig. 11.16) we compared with include multi-layer perceptron on unsorted (with permutation augmentation) and sorted (with lex-sort based on XYZ coordinates) points as $n \times 3$ arrays, an RNN model that considers input points as a sequence, and PointNet models (vanilla versions without input and feature transformations) based on symmetry functions. The symmetry operation that we experimented with include max pooling, average pooling and an attention-based weighted sum. The attention method is similar to that in [103], where a scalar score is predicted from each point feature, then the score is normalized across points by computing a softmax. The weighted sum is then computed on the normalized scores and the point features. As shown in Fig. 11.16, max-pooling operation achieved the best performance by a large margin, which validates the PointNet's design choice.



	Accuracy
MLP (unsorted input)	24.2
MLP (sorted input)	45.0
LSTM	78.5
Attention sum	83.0
Average pooling	83.8
Max pooling	87.1

Fig. 11.16 Three approaches to achieve order invariance. Multi-layer perceptron (MLP) applied on points consists of 5 hidden layers with neuron sizes 64, 64, 64, 128, 1024, all points share a single copy of MLP. The MLP close to the output consists of two layers with sizes 512, 256. Figure is from [76]

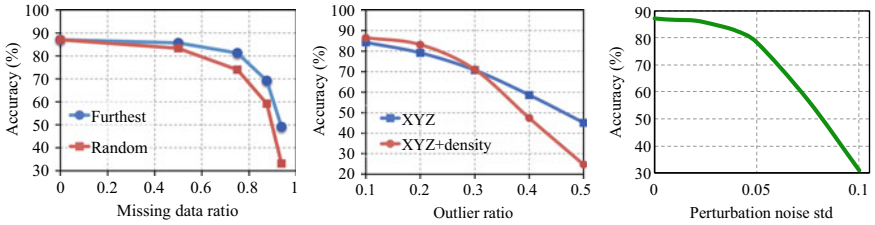


Fig. 11.17 PointNet robustness test. The metric is overall classification accuracy on ModelNet40 test set. Left: delete points. Furthest means the original 1024 points are sampled with furthest sampling. Middle: insertion. Outliers uniformly scattered in the unit sphere. Right: perturbation. Add Gaussian noise to each point independently. Figure is from [76]

Robustness Test

We show that PointNet, while simple and effective, is robust to various kinds of input corruptions. We use the same architecture as in Fig. 11.16's max-pooling network. Input points are normalized into a unit sphere. Results are in Fig. 11.17.

As to missing points, when there are 50% points missing, the accuracy only drops by 2.4 and 3.8% with respect to furthest and random input sampling. The network is also robust to outlier points, if it has seen those during training. We evaluate two models: one trained on points with (x, y, z) coordinates; the other on (x, y, z) plus point density. The network has more than 80% accuracy even when 20% of the points are outliers. Figure 11.17 (right) shows the network is robust to point perturbations.

We further compare PointNet and VoxNet [65] on robustness to missing data in the input point clouds. Both networks are trained with the same train-test split, using 1024 points per shape as input. For VoxNet we voxelize the point cloud to $32 \times 32 \times 32$ occupancy grids and augment the training data by random rotation around the up-axis and jittering.

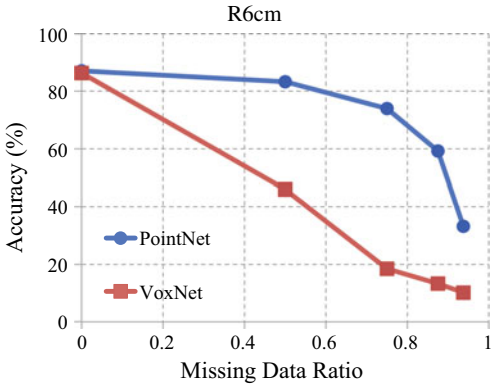


Fig. 11.18 PointNet versus VoxNet [65] on incomplete input data. Metric is overall classification accuracy on ModelNet40 test set. Note that VoxNet is using 12 viewpoints averaging while PointNet is using only one view of the point cloud. Evidently PointNet presents much stronger robustness to missing points. Figure is from [76]

At test time, input points are randomly dropped out by a certain ratio. As VoxNet is sensitive to rotations, its prediction uses average scores from 12 viewpoints of a point cloud. As shown in Fig. 11.18, we see that the PointNet is much more robust to missing points. VoxNet's accuracy dramatically drops when half of the input points are missing, from 86.3 to 46.0% with a 40.3% difference, which is more than *10x* worse than PointNet (only 3.8% drop in accuracy). This can be explained by the theoretical analysis and explanation of PointNet—it learns to use a collection of *critical points* to summarize the shape, thus it is very robust to missing data.

11.4.5.3 Visualizing PointNet

Critical Point and Upper-Bound Shape Visualization

In Fig. 11.19, we visualize some results of the *critical point sets* $\mathcal{C}_S$ and the *upper-bound shapes* $\mathcal{N}_S$ (as discussed in Theorem 11.2) for some sample shapes S. The point sets between the two shapes will give exactly the same global shape feature $f(S)$.

We can see clearly from Fig. 11.19 that the *critical point sets* $\mathcal{C}_S$, contributed to the max pooled feature, summarizes the skeleton of the shape. The *upper-bound shapes* $\mathcal{N}_S$ illustrates the largest possible point cloud that gives the same global shape feature $f(S)$ as the input point cloud S. $\mathcal{C}_S$ and $\mathcal{N}_S$ reflect the robustness of PointNet, meaning that losing some non-critical points does not change the global shape signature $f(S)$ at all.

The $\mathcal{C}_S$ is computed by comparing point embedding values, i.e., $h(x_i)$ with the max-pooled embedding for each embedding dimension and each point: if a point's embedding is smaller than the max-pooled embedding in all dimensions, then this point does not contribute to the pooled descriptor and thus is not a critical point; otherwise it is a critical point. The $\mathcal{N}_S$ is constructed (approximately) by forwarding a uniformly and densely sampled set of points in $[-1, 1]^3$ through the network and

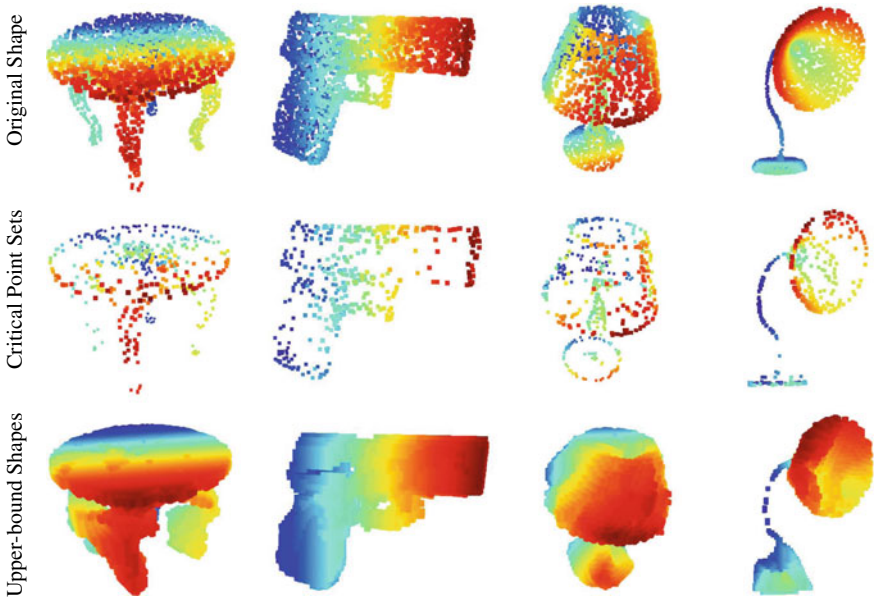


Fig. 11.19 Critical points and upper-bound shape. While critical points jointly determine the global shape feature for a given shape, any point cloud that falls between the critical points set and the upper-bound shape gives exactly the same feature. We color-code all figures to show the depth information. Figure is from [76]

select points p whose point function values $(h_1(p), h_2(p), \ldots, h_K(p))$ are no larger than the global shape descriptor.

Point Function Visualization

The classification PointNet computes a K-dimensional ($K = 1024$ in this visualization) point embedding for each point and aggregates all the per-point embeddings via a max-pooling layer into a single K-dim vector, which forms a global shape descriptor.

To gain more insights on what the point functions $h_1, \ldots, h_K$ have learned, we visualize the points x_i's that give high point function value $h(x_i)$ in Fig. 11.20. This visualization clearly shows that different point functions learn to detect points in different regions with various shapes scattered in the whole space.

In another perspective, the point function h_i can be viewed as a **space encoding function** for input space $\mathbb{R}^N$. To understand this view, we can take voxelization as an example. Voxelization is a predefined, static way to encode space. For example, in a $10 \times 10 \times 10$ binary occupancy grid of the $\mathbb{R}^3$ space, there are 1000 encoding functions corresponding to the 1000 voxels, which are all one-hot functions mapping from $\mathbb{R}^3$ to $\{0, 1\}$ with 0 noting an empty voxel and 1 noting an occupied one—the

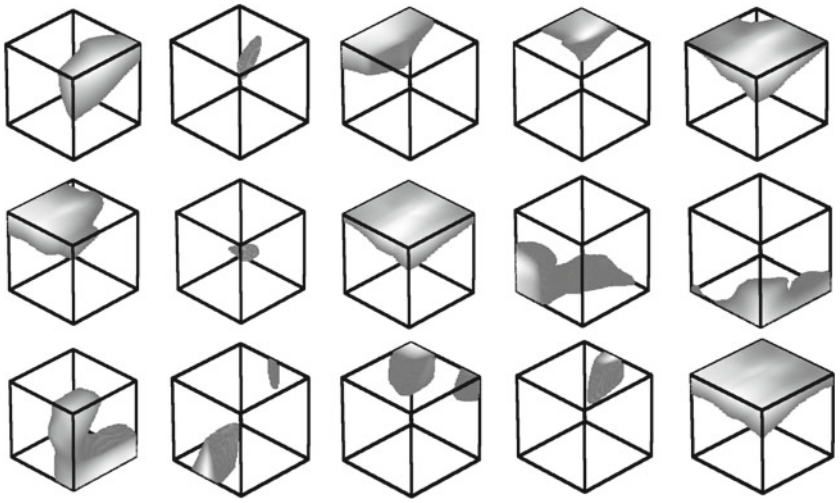


Fig. 11.20 Point function visualization. For each per-point function h, we calculate the values $h(p)$ for all the points p in a cube of diameter two located at the origin, which spatially covers the unit sphere to which our input shapes are normalized when training our PointNet. In this figure, we visualize all the points p that give $h(p) > 0.5$ with function values color-coded by the brightness of the voxel. We randomly pick 15 point functions and visualize the activation regions for them

ith function is activated only if there is a point in the ith voxel. In comparison, in PointNet, we are able to learn a set of smooth space encoding functions adaptive to data and task, which could encode the space in a more efficient and effective way than the simple voxelization.

11.4.5.4 Time and Space Complexity Analysis

Table 11.5 summarizes space (number of parameters in the network) and time (floating-point operations/sample) complexity of the classification PointNet. We also compare PointNet to a representative set of volumetric and multi-view based architectures in previous works.

While MVCNN [92] and Subvolume (3D CNN) [75] achieve high performance, PointNet is orders more efficient in computational cost (measured in FLOPS: *141x* and *8x* more efficient, respectively). Besides, PointNet is much more space efficient than MVCNN in terms of the number of parameters in the network (*17x* less parameters). Moreover, PointNet is much more scalable—its space and time complexity is $O(N)$-*linear* in the number of input points. However, since convolution dominates computing time, multi-view method's time complexity grows *quadratically* on image resolution and volumetric convolution-based method grows *cubically* with the volume size.

Table 11.5 Time and space complexity of deep architectures for 3D data classification. PointNet (vanilla) is the classification PointNet without input and feature transformations. FLOPS stands for floating-point operations per second. The "M" stands for million. Subvolume and MVCNN used pooling on input data from multiple rotations or views, without which they have much inferior performance

	Number of parameters	FLOPS
PointNet (vanilla) [76]	0.8M	148M
PointNet [76]	3.5M	440M
Subvolume [75]	16.6M	3633M
MVCNN [92]	60.0M	62057M

Empirically, PointNet is able to process more than one million points per second for point cloud classification (around 1 K objects/second) or semantic segmentation (around 2 rooms/second) with a 1080X GPU on TensorFlow, showing great potential for portable devices and real-time applications.

11.5 Deep Learning on Meshes

A polygon mesh is a collection of vertices, edges and faces that define the shape of a 3D object in computer graphics or solid modeling. Each vertex may be associated with extra data than its coordinates such as its local geometry property (such as normal and curvature) or other physical properties (such as surface tension and temperature). In the context of data structures, a polygon mesh is a *graph* thus the deep neural networks processing it are often called *graph CNNs* or *geometric deep learning* models, which are generalizations of convolutional neural networks defined on a Euclidean or grid-like structure to a non-Euclidean domain [11]. The architectures studied under geometric deep learning are not restricted to analyzing 3D objects or scenes, they can be used in much broader data types such as social networks (with users as vertices), sensor networks (with distributed interconnected sensors as vertices), or even gene expression data and brain structure data that are often modeled as graphs [11].

The methods of graph CNNs mainly fall into one of the following two categories. The first category of methods is based on the Convolution Theorem, to define convolution in the *spectral* domain [12, 21, 38, 115]. The main motivation behind this formulation is because there is no obvious definition of translation in the spatial domain. This branch of methods is tightly related to the field of spectral analysis on graphs. An alternative family of methods considers the convolution operation as a template matching process in the *spatial* domain [8, 9, 64, 68], and focuses on defining spatial operators on a local graph. However, as noted in [11], there is often not a clear cut between spectral and spatial methods. Aside from these two families of methods, there are also methods that convolve the graphs in an embedding

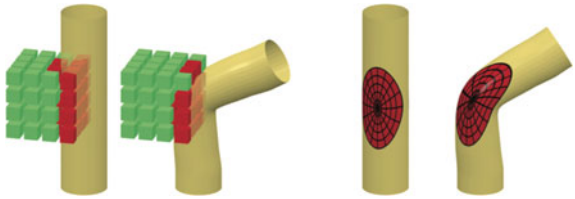


Fig. 11.21 Illustration of the difference between extrinsic and intrinsic deep learning methods on geometric data. Left: extrinsic methods. Right: intrinsic methods. Figure is from [9]

space mapped (such as a parameterization space in 2D) from the original mesh and resort to traditional CNNs for feature learning [63]. However, in those cases the parameterization is often not unique and introduces distortion to the data.

In the following text, we briefly introduce the design ideas and application results behind two representative works of deep learning on meshes, one based on the spatial domain (ACNN [9]) and the other on the spectral domain (SyncSpecCNN [115]).

11.5.1 Spatial Domain Graph CNN

While graph CNNs are also called CNNs, they have significant differences with image CNNs and volumetric CNNs defined on Euclidean space/regular grid structures. Figure 11.21 gives a clear illustration of their difference. On the left of the figure, we see that volumetric CNNs define the convolution kernels, as well as translations in the 3D Euclidean space. Such an operation is not invariant to deformations. For example as seen in the figure the filter activated from a straight cylinder would not respond if the tube is bent. In comparison, on the right we see an intrinsic convolution operation, where the receptive field is defined on the surface itself, thus invariant to deformations. For deformable objects such as human bodies, the ability to respect deformation invariance is critical to achieve good performance in tasks like finding correspondences.

In the work of the ACNN (anisotropic convolutional neural network) [9], a set of oriented anisotropic diffusion kernels are constructed which operate on the local intrinsic polar representation of the data. With several cascades of the filters using both linear and nonlinear layers, a deep neural network is trained to learn intrinsic dense correspondences between deformable human bodies. Figure 11.22 shows the remarkable results achieved by the method.

11.5.2 Spectral Domain Graph CNN

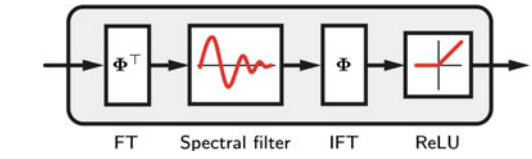
In spectral methods of graph CNNs, a 3D shape S is modeled as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with vertices $\mathcal{V}$ as points in $\mathbb{R}^3$ and $|\mathcal{V}| = n$, and edges $\mathcal{E}$ as the set of edges in the graph. The adjacency matrix associated with $\mathcal{G}$ is denoted as $W \in \mathbb{R}^{n \times n}$. One way

Fig. 11.22 Examples of correspondence on the FAUST humans dataset [7] obtained by ACNN. Shown is the texture transferred from the leftmost reference shape to different subjects in different poses by means of predicted correspondence. The correspondence is nearly perfect (only very few minor artifacts are noticeable). Figure is from [9]

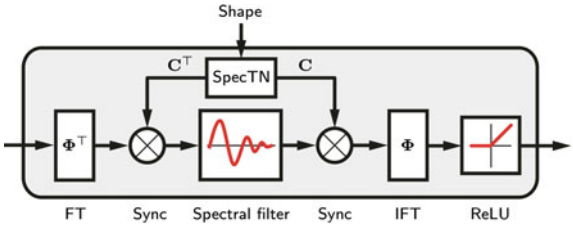
to compute the Laplacian of the graph is $L = I - D^{-1/2}WD^{-1/2}$ where I is the identify matrix, and D is the degree matrix where $D_{i,i} = \sum_j W_{i,j}$. The eigenvalue decomposition gives a set of eigenvectors $u_l, l = 0, \ldots, n-1$ as a set of orthogonal bases of the graph along with their corresponding eigenvalues $\lambda_l, l = 0, \ldots, n-1$. As in Fourier analysis, this spectral decomposition also introduces the concept of frequency. For each basis u_l, the eigenvalue λ_l in the decomposition defines its frequency, depicting its smoothness. The convolution theorem of Fourier analysis can be extended to the Laplacian spectrum: the convolution between a kernel and a function on the shape graph is equivalent to the pointwise multiplication of their spectral representations [12].

In the work of SyncSpecCNN [115], the graph CNN is defined based on the spectral convolution theorem. It also combines processing in the spectral and spatial domains, where weight sharing among convolution kernels at different scales is by performing the convolutions in the spectral domain (as pointwise multiplications by the kernel duals), while the nonlinearity step (such operations are not easily dualized) happens in the spatial domain. The general process is illustrated in Fig. 11.23a.

One challenge of this spectral based method is on how to effectively share weights across shapes. Since different shapes give rise to different nearest neighbor graphs on their point clouds, the eigenbases we get for the graph Laplacians are not directly comparable. To address the issue in SyncSpecCNN, the authors proposed to synchronize all these Laplacians by applying a functional map [70] (a linear map between function spaces) in the spectral domain to align them to a common canonical space. The aligning functional maps succeed in encoding all the dual information on a common set of basis functions where global learning takes place. An initial version of the aligning maps is computed directly from the geometry and then is further refined during training, in the style of a data-dependent spatial transformer network. The newly proposed spectral convolution scheme is illustrated in Fig. 11.23b. With the ability to sync eigenbases across geometrically very different shapes, the SyncSpecCNN is able to generalize to fine-grained tasks on man-made objects, which



(a) Spectral convolution.



(b) Synced spectral convolution.

Fig. 11.23 Comparisons of spectral convolution and synced spectral convolution. Figure adapted from [40]

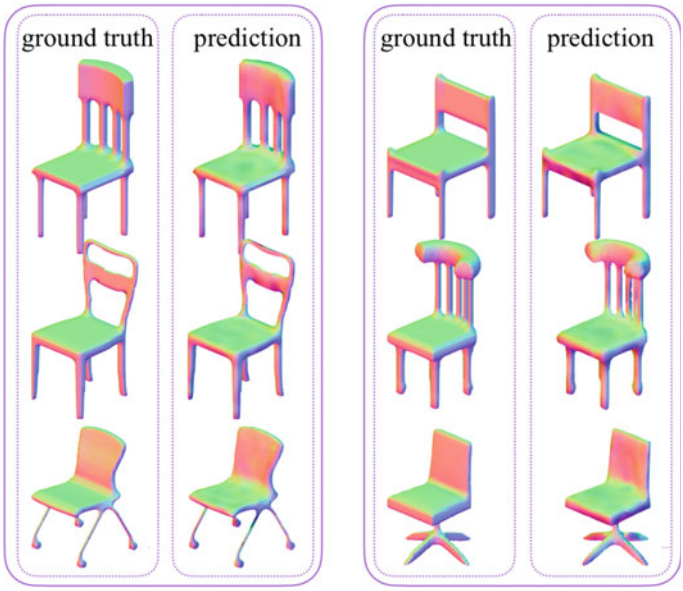


Fig. 11.24 Normal prediction task. The colors shown on the 3D shape are RGB-coded normals, namely putting XYZ components of normal directions into RGB channels. The SyncSpecCNN predicts reasonable normal directions even on very thin structures. Figure is from [115]

was previously hard for spectral graph CNN methods. Figure 11.24 demonstrates its successful normal predictions.

11.6 Conclusion and Outlook

This chapter has presented a rich family of methods on deep 3D representation learning. Starting from deep learning models on regularly structured 3D data, we showed how we can represent 3D shapes as multi-view images and how we can leverage existing CNNs to build a deep network to learn to aggregate information from those views to produce a powerful 3D descriptor. We presented a simple yet effective architecture for processing volumetric input. We discussed the reasons behind the performance gap between volumetric CNNs and multi-view CNNs, while also introducing two new volumetric architectures to fill the gap. Then in the next two sections, we presented deep learning models on irregular data: point clouds and meshes. A permutation invariant network PointNet is described in detail with both theoretical analysis and rich experimental and visualization results. Finally, we introduced two main families of deep networks for processing meshes.

Due to the high reproducibility of deep learning-based methods, now there is a trend in the general deep learning community to open source model architectures and experiment scripts. In fact, most methods presented in this paper have open sourced code and datasets from the original authors.[7] We believe such code releases encourage more reproducible research and to accelerates the development cycles of future work. We encourage readers interested in the details of the methods to inspect such code and run their experiments.

Future Work

3D deep learning is still a young field, there are many more interesting future directions for research. One such direction is to develop architectures that further respect rotation invariance or even achieve rotation equivariance (a rotated point cloud leads to "rotated" features). We have seen naive ways to address the issue through pooling (MVCNN and multi-orientation pooling in volumetric CNNs) or data augmentation, or input spatial transformer networks. However, they are not designed to guarantee any invariance or equivariance in rotations. Some very recent works try to address the problem with spherical CNNs [17, 25], but they are restricted to single objects and are not applicable to 3D scenes. In future work, it is worth exploring more generic architectures equivariant to rotations, for example as 3D variants of Capsule Networks [83].

Another timely research direction is joint learning of geometry and color data. In this chapter we focused on the geometry data alone, but real applications usually get access to both point clouds and 2D images. These two data sources are often complementary as point clouds describe accurate 3D geometry (but low in resolution)

[7] MVCNN: https://github.com/suhangpro/mvcnn.
VoxNet: https://github.com/dimatura/voxnet.
PointNet: http://github.com/charlesq34/pointnet.
SyncSpecCNN: https://github.com/ericyi/SyncSpecCNN.

and images capture high-resolution color textures (but lack 3D). We have already touched on some combined use cases of the two modalities. For example in PointNet semantic segmentation, the best results are achieved when we append RGB color as extra feature channels for each point. However, this is still an early attempt in this direction. More recently there are more efforts in joining 2D and 3D features for semantic segmentation [93, 97], as well as in 3D object detection [53, 112]. It is still interesting to study different design choices: whether we should aggregate both 2D and 3D features on 2D, on 3D or in another latent space, in which we need more research.

Last but not least, besides *analyzing* 3D data such as volumetric grids and point clouds, it is very fascinating to study how we can *generate* or *synthesize* high quality 3D geometry, for 3D reconstruction, shape completion or AI-assisted design/modeling. In image generation, a generative model only needs to decide pixel values since all pixel locations are already fixed. In comparison, to generate a point cloud, a model has to decide positions for points, which could be a much more challenging problem. Fan et al. [27] is one of the early works in this direction, where they reconstruct 3D geometry (with point clouds) from a single image. Another work called Octree generation network [96] manages to generate high-resolution 3D with Octree structures. Despite those early works, how to generate 3D point clouds with richer features such as position confidence and color, and how to deal with ambiguity in the generated space are still very open problems.

Broad Applications of Deep Learning on 3D Data

3D data is prevalent and how to process it is a fundamental problem. So its applications are also very diverse and are beyond the ones we mentioned in this chapter.

For example, AI-assisted design is a rising field where designers work with machine learning algorithms to accelerate the work cycle and improve overall design quality. In a project called ComplementMe [95], an algorithm can provide interactive design assistance to a 3D shape designer. The system provides suggestions to the designer in real time, on possible shape parts to use, as well as where to place them so that they are compatible with existing parts. In this system, PointNet is used to learn a latent representation for shape parts. Deep architectures on point clouds can also complement other deep networks in learning more robust 3D shape descriptors, for building large-scale 3D shape search engines. In AtlasNet [34], 3D deep learning models help model 3D meshes from simple raw point clouds or images as input.

Another promising field to apply 3D deep learning is robotics. In many robotics tasks such as grasping and manipulation, it is critical to get robust semantic understanding and accurate localization of 3D geometry. Point clouds are the prevalent sensor data in those tasks. A light-weight 3D deep learning architecture would provide a new machinery to directly process point clouds and understand their semantics and geometry. 3D deep learning models can also be used for learning intuitive physics, where we infer physical rules or properties by observing objects move. Direct learn-

ing in point clouds can potentially simplify the problem compared to learning in images.

The presented networks and future 3D deep learning models also have a potential impact on more general science. One interesting problem is to infer functions from structures in molecules, which can be formulated as a learning problem on point clouds. For example, given 3D arrangement of atoms in enzymes, machines can be trained to classify enzymes, which can potentially accelerate scientific discoveries and save labor costs. Another direction is to process and understand medical images, especially when the signals are sparse and represented as point clouds, in which successful algorithms could save lives.

We believe this chapter just marks the beginning of the exciting and fast growing field of 3D deep learning. We expect to see more exciting research in the future.

11.7 Further Reading

Lots of related resources on 3D deep learning are still in the forms of articles, courses, and conference tutorials. For a more broad overview of the topics mentioned in this chapter, we refer the reader to the 3D deep learning tutorial from CVPR 2017 [40], which covers more applications and example architectures. We also refer readers to two relevant University courses with notes available online: *Machine Learning for 3D Data* from Stanford University [42] and a similar course with more focus on deep learning methods from University of California San Diego [43]. While focusing on multi-view, volumetric, and point cloud representations, we put less content on deep learning models for meshes because it falls in another large research topic—deep learning on graphs. For readers with more interests in graph/geometric deep learning, we refer them to a very thorough review paper from Bronstein et al. [11]. Details of the ShapeNet dataset and the motivation and vision behind this effort can be found in [14]. For a detailed literature explaining how deep learning works, we recommend the reader to the *Deep Learning* textbook [31] and the Stanford course [41] while the latter has a particular focus on the convolutional neural networks.

11.8 Questions

1. Give three examples of applications of deep learning on 3D data.
2. Explain the challenges of applying convolutional neural networks on point clouds and meshes. What are some common ways to convert them such that we can use CNNs?
3. What are the pro and cons of multi-view CNNs?
4. What are the pro and cons of volumetric CNNs?
5. What are the possible reasons that a volumetric CNN lags behind in 3D object classification performance compared with a multi-view CNN?

6. What qualities do we look for in a deep learning model consuming raw point clouds?
7. What are the advantages of a PointNet model? Are there any limitations?
8. What are the two common ways to achieve deep learning on meshes?
9. What is the difference between extrinsic convolutions on regular grids and intrinsic convolutions on surfaces?
10. List three future research directions for deep learning on 3D data.

11.9 Exercises

1. Draw a t-SNE [102] visualization from 3D descriptors extracted by a trained MVCNN on a collection of 3D shapes (e.g., from ModelNet40 [110]). Do you observe that shapes from the same category are embedded close to each other? Can you also find some confusing cases from the visualization?
2. Given a trained MVCNN, test how the number of input images affects results. Assuming we know the alignment of objects, does a certain viewpoint (e.g., the front view of the aligned objects) matter more than the back view?
3. Given a CAD model, voxelize it to binary occupancy grids in different resolutions (using the hollow model), e.g., to $10 \times 10 \times 10$, $30 \times 30 \times 30$ and $60 \times 60 \times 60$. Observe how occupancy rate (the ratio between the number of the occupied voxels and that of all the voxels) decreases as we increase resolution.
4. Implement a sphere rendering engine that renders 2D views from point clouds. See if you can tell the category of an object from its low resolution sphere rendering. How many points do you need approximately to tell the correct categories?
5. Experiment with a VoxNet that takes truncated signed distance field (TSDF) or truncated distance field (TDF) grids as input. Do the TSDF or TDF features lead to a stronger classification performance compared with binary occupancy feature (e.g., on the ModelNet40 benchmark)?
6. Download a trained PointNet model and the ModelNet40 dataset, and then visualize the top-10 critical points from the PointNet on several shapes. Do they have patterns or consistency among shapes from the same category?
7. Implement a point cloud autoencoder using a PointNet architecture for the encoder and a fully connected layer for the decoder. Refer to [26] on how to define an optimization loss to compare two point clouds.
8. In 3D deep learning, there is often a need to code in GPU to achieve fast parallel layers for geometric processing. Sharpen your GPU coding skills by trying to implement the farthest point sampling in CUDA.
9. In [117], the authors proposed a 3D deep learning model with mixed representations. It first learns voxel features from local point clouds in each voxel and then uses 3D CNNs for voxel feature learning. Implement the VoxelNet structure for ModelNet40 object classification, and try to experiment with the hyperparameters such as grid resolution, network depth, and widths. Compare results with a VoxNet that is based on 3D CNN only.

10. Download the KITTI object detection dataset and extract the point clouds within all the ground truth 3D object bounding boxes. Train a PointNet to classify the LiDAR point clouds into different semantic categories (pedestrian, cars, and cyclists).
11. Implement a spatial method for a graph convolution network and test it on object part segmentation on ShapeNet objects. Refer to [11] to find a spatial graph CNN model you like. Compare results with those of the method in [115].

References

1. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al.: TensorFlow: a system for large-scale machine learning. In: OSDI, vol. 16, pp. 265–283 (2016)
2. Amodei, D., Ananthanarayanan, S., Anubhai, R., Bai, J., Battenberg, E., Case, C., Casper, J., Catanzaro, B., Cheng, Q., Chen, G., et al.: Deep speech 2: end-to-end speech recognition in English and Mandarin. In: International Conference on Machine Learning, pp. 173–182 (2016)
3. Armeni, I., Sener, O., Zamir, A.R., Jiang, H., Brilakis, I., Fischer, M., Savarese, S.: 3D semantic parsing of large-scale indoor spaces. In: Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (2016)
4. Aubry, M., Schlickewei, U., Cremers, D.: The wave kernel signature: a quantum mechanical approach to shape analysis. In: 2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops), pp. 1626–1633. IEEE (2011)
5. Belongie, S., Malik, J., Puzicha, J.: Shape context: a new descriptor for shape matching and object recognition. In: Advances in Neural Information Processing Systems, pp. 831–837 (2001)
6. Belton, D., Lichti, D.D.: Classification and segmentation of terrestrial laser scanner point clouds using local variance information. IAPRS **Xxxvi**(5), 44–49 (2006)
7. Bogo, F., Romero, J., Loper, M., Black, M.J.: FAUST: dataset and evaluation for 3D mesh registration. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 3794–3801 (2014)
8. Boscaini, D., Masci, J., Melzi, S., Bronstein, M.M., Castellani, U., Vandergheynst, P.: Learning class-specific descriptors for deformable shapes using localized spectral convolutional networks. Computer Graphics Forum, vol. 34, pp. 13–23. Wiley Online Library, Hoboken (2015)
9. Boscaini, D., Masci, J., Rodolà, E., Bronstein, M.: Learning shape correspondence with anisotropic convolutional neural networks. In: Advances in Neural Information Processing Systems, pp. 3189–3197 (2016)
10. Bronstein, M.M., Kokkinos, I.: Scale-invariant heat kernel signatures for non-rigid shape recognition. In: 2010 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1704–1711. IEEE (2010)
11. Bronstein, M.M., Bruna, J., LeCun, Y., Szlam, A., Vandergheynst, P.: Geometric deep learning: going beyond Euclidean data. IEEE Signal Process. Mag. **34**(4), 18–42 (2017)
12. Bruna, J., Zaremba, W., Szlam, A., LeCun, Y.: Spectral networks and locally connected networks on graphs (2013). arXiv:1312.6203
13. Chang, A., Dai, A., Funkhouser, T., Halber, M., Niessner, M., Savva, M., Song, S., Zeng, A., Zhang, Y.: Matterport3D: learning from RGB-D data in indoor environments. In: International Conference on 3D Vision (3DV) (2017)

14. Chang, A.X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M., Song, S., Su, H., Xiao, J., Yi, L., Yu, F.: ShapeNet: an information-rich 3D model repository. Technical report, Stanford University — Princeton University — Toyota Technological Institute at Chicago (2015). arXiv:1512.03012 [cs.GR]
15. Chen, D.Y., Tian, X.P., Shen, Y.T., Ouhyoung, M.: On visual similarity based 3D model retrieval. Computer Graphics Forum, vol. 22, pp. 223–232. Wiley Online Library, Hoboken (2003)
16. Choy, C., Gwak, J., Savarese, S.: 4D spatio-temporal ConvNets: Minkowski convolutional neural networks. (2019). arXiv:1904.08755
17. Cohen, T.S., Geiger, M., Köhler, J., Welling, M.: Spherical CNNs (2018). arXiv:1801.10130
18. Dai, A., Nießner, M.: 3DMV: joint 3D-multi-view prediction for 3D semantic scene segmentation. In: Proceedings of the European Conference on Computer Vision (ECCV), pp. 452–468 (2018)
19. Dai, A., Chang, A.X., Savva, M., Halber, M., Funkhouser, T.A., Nießner, M.: ScanNet: richly-annotated 3D reconstructions of indoor scenes. In: CVPR, vol. 2, p. 10 (2017)
20. Dai, A., Ruizhongtai Qi, C., Nießner, M.: Shape completion using 3D-encoder-predictor CNNs and shape synthesis. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 5868–5877 (2017)
21. Defferrard, M., Bresson, X., Vandergheynst, P.: Convolutional neural networks on graphs with fast localized spectral filtering. In: Advances in Neural Information Processing Systems, pp. 3844–3852 (2016)
22. Demantké, J., Mallet, C., David, N., Vallet, B.: Dimensionality based scale selection in 3D lidar point clouds. Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci. **38**(Part 5), W12 (2011)
23. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: ImageNet: a large-scale hierarchical image database. In: IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2009, pp. 248–255. IEEE (2009)
24. Duvenaud, D.K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., Adams, R.P.: Convolutional networks on graphs for learning molecular fingerprints. In: Advances in Neural Information Processing Systems, pp. 2224–2232 (2015)
25. Esteves, C., Allen-Blanchette, C., Makadia, A., Daniilidis, K.: Learning so (3) equivariant representations with spherical CNNs. In: ECCV 2018 (2018)
26. Fan, H., Su, H., Guibas, L.J.: A point set generation network for 3D object reconstruction from a single image. In: CVPR, vol. 2, p. 6 (2017)
27. Fan, H., Su, H., Guibas, L.J.: A point set generation network for 3D object reconstruction from a single image. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2017)
28. Fang, Y., Xie, J., Dai, G., Wang, M., Zhu, F., Xu, T., Wong, E.: 3D deep shape descriptor. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2319–2328 (2015)
29. Geiger, A., Lenz, P., Stiller, C., Urtasun, R.: Vision meets robotics: the KITTI dataset. Int. J. Robot. Res. **32**(11), 1231–1237 (2013)
30. Girshick, R.: Fast R-CNN. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 1440–1448 (2015)
31. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press, Cambridge (2016). http://www.deeplearningbook.org
32. Graham, B., Engelcke, M., van der Maaten, L.: 3D semantic segmentation with submanifold sparse convolutional networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 9224–9232 (2018)
33. Gressin, A., Mallet, C., Demantké, J., David, N.: Towards 3D lidar point cloud registration improvement using optimal neighborhood knowledge. ISPRS J. Photogramm. Remote Sens. **79**, 240–251 (2013)
34. Groueix, T., Fisher, M., Kim, V.G., Russell, B.C., Aubry, M.: AtlasNet: a Papier-Mâché approach to learning 3D surface generation (2018). arXiv:1802.05384

35. Gschwandtner, M., Kwitt, R., Uhl, A., Pree, W.: BlenSor: blender sensor simulation toolbox. Advances in Visual Computing, pp. 199–208. Springer, Berlin (2011). https://doi.org/10.1007/978-3-642-24031-7_20
36. Hackel, T., Savinov, N., Ladicky, L., Wegner, J.D., Schindler, K., Pollefeys, M.: Semantic3D.net: a new large-scale point cloud classification benchmark. ISPRS Ann. Photogramm. Remote Sens. Spat. Inf. Sci. **IV-1-W1**, 91–98 (2017)
37. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)
38. Henaff, M., Bruna, J., LeCun, Y.: Deep convolutional networks on graph-structured data (2015). arXiv:1506.05163
39. Hermosilla, P., Ritschel, T., Vázquez, P.P., Vinacua, À., Ropinski, T.: Monte Carlo convolution for learning on non-uniformly sampled point clouds. In: SIGGRAPH Asia 2018 Technical Papers, p. 235. ACM (2018)
40. Homepage of 3D deep learning tutorial at CVPR 2017, Honolulu, United States. http://3ddl.stanford.edu. Accessed 25 Oct 2018
41. Homepage of CS231N: Convolutional neural networks for visual recognition at Stanford University. http://cs231n.stanford.edu. Accessed 25 Oct 2018
42. Homepage of CS468: Machine learning for 3D data at Stanford University. http://graphics.stanford.edu/courses/cs468-17-spring/. Accessed 25 Oct 2018
43. Homepage of CS468: Machine learning for 3D data at University of California San Diego. https://cse291-i.github.io. Accessed 25 Oct 2018
44. Hua, B.S., Tran, M.K., Yeung, S.K.: Pointwise convolutional neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 984–993 (2018)
45. Jaderberg, M., Simonyan, K., Zisserman, A., et al.: Spatial transformer networks. In: NIPS 2015
46. Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., Darrell, T.: Caffe: convolutional architecture for fast feature embedding (2014). arXiv:1408.5093
47. Johnson, A.E., Hebert, M.: Using spin images for efficient object recognition in cluttered 3D scenes. IEEE Trans. Pattern Anal. Mach. Intell. **21**(5), 433–449 (1999)
48. Kalogerakis, E., Averkiou, M., Maji, S., Chaudhuri, S.: 3D shape segmentation with projective convolutional networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 3779–3788 (2017)
49. Kato, H., Ushiku, Y., Harada, T.: Neural 3D mesh renderer. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 3907–3916 (2018)
50. Kazhdan, M., Funkhouser, T., Rusinkiewicz, S.: Rotation invariant spherical harmonic representation of 3D shape descriptors. In: Symposium on Geometry Processing, vol. 6, pp. 156–164 (2003)
51. Klokov, R., Lempitsky, V.: Escape from cells: deep Kd-networks for the recognition of 3D point cloud models. In: 2017 IEEE International Conference on Computer Vision (ICCV), pp. 863–872. IEEE (2017)
52. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems, pp. 1097–1105 (2012)
53. Ku, J., Mozifian, M., Lee, J., Harakeh, A., Waslander, S.: Joint 3D proposal generation and object detection from view aggregation (2017). arXiv:1712.02294
54. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436–444 (2015)
55. Li, J., Chen, B.M., Hee Lee, G.: SO-Net: self-organizing network for point cloud analysis. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 9397–9406 (2018)
56. Li, Y., Su, H., Qi, C.R., Fish, N., Cohen-Or, D., Guibas, L.J.: Joint embeddings of shapes and images via CNN image purification. TOG (2015)
57. Li, Y., Bu, R., Sun, M., Wu, W., Di, X., Chen, B.: PointCNN: convolution on X-transformed points. In: Advances in Neural Information Processing Systems, pp. 820–830 (2018)

58. Lian, Z., Zhang, J., Choi, S., ElNaghy, H., El-Sana, J., Furuya, T., Giachetti, A., Guler, R.A., Lai, L., Li, C., Li, H., Limberger, F.A., Martin, R., Nakanishi, R.U., Neto, A.P., Nonato, L.G., Ohbuchi, R., Pevzner, K., Pickup, D., Rosin, P., Sharf, A., Sun, L., Sun, X., Tari, S., Unal, G., Wilson, R.C.: Non-rigid 3D shape retrieval. In: Pratikakis, I., Spagnuolo, M., Theoharis, T., Gool, L.V., Veltkamp, R. (eds.) Eurographics Workshop on 3D Object Retrieval. The Eurographics Association (2015). https://doi.org/10.2312/3dor.20151064
59. Lin, M., Chen, Q., Yan, S.: Network in network (2013). arXiv:1312.4400
60. Ling, H., Jacobs, D.W.: Shape classification using the inner-distance. IEEE Trans. Pattern Anal. Mach. Intell. **29**(2), 286–299 (2007)
61. Loper, M.M., Black, M.J.: OpenDR: an approximate differentiable renderer. In: European Conference on Computer Vision, pp. 154–169. Springer (2014)
62. Maas, A.L., Hannun, A.Y., Ng, A.Y.: Rectifier nonlinearities improve neural network acoustic models. In: Proceedings of the ICML, vol. 30, p. 3 (2013)
63. Maron, H., Galun, M., Aigerman, N., Trope, M., Dym, N., Yumer, E., Kim, V.G., Lipman, Y.: Convolutional neural networks on surfaces via seamless toric covers. ACM Trans. Graph. **36**(4), 71 (2017)
64. Masci, J., Boscaini, D., Bronstein, M., Vandergheynst, P.: Geodesic convolutional neural networks on Riemannian manifolds. In: Proceedings of the IEEE International Conference on Computer Vision Workshops, pp. 37–45 (2015)
65. Maturana, D., Scherer, S.: VoxNet: a 3D convolutional neural network for real-time object recognition. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (2015)
66. Mikolov, T., Deoras, A., Povey, D., Burget, L., Černockỳ, J.: Strategies for training large scale neural network language models. In: 2011 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU), pp. 196–201. IEEE (2011)
67. Mitra, N.J., Nguyen, A., Guibas, L.: Estimating surface normals in noisy point cloud data. Int. J. Comput. Geom. Appl. **14**(04–05), 261–276 (2004)
68. Monti, F., Boscaini, D., Masci, J., Rodola, E., Svoboda, J., Bronstein, M.M.: Geometric deep learning on graphs and manifolds using mixture model CNNs. In: Proceedings of the CVPR, vol. 1, p. 3 (2017)
69. Osada, R., Funkhouser, T., Chazelle, B., Dobkin, D.: Shape distributions. ACM Trans. Graph. (TOG) **21**(4), 807–832 (2002)
70. Ovsjanikov, M., Ben-Chen, M., Solomon, J., Butscher, A., Guibas, L.: Functional maps: a flexible representation of maps between shapes. ACM Trans. Graph. (TOG) **31**(4), 30 (2012)
71. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in PyTorch. In: NIPS-W (2017)
72. Pauly, M., Kobbelt, L.P., Gross, M.: Point-based multiscale surface representation. ACM Trans. Graph. (TOG) **25**(2), 177–193 (2006)
73. Phong, B.T.: Illumination for computer generated pictures. Commun. ACM **18**(6), 311–317 (1975)
74. Qi, R.: Deep learning on point clouds for 3D scene understanding. Ph.D. thesis, Stanford University (2018)
75. Qi, C.R., Su, H., Nießner, M., Dai, A., Yan, M., Guibas, L.: Volumetric and multi-view CNNs for object classification on 3D data. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2016)
76. Qi, C.R., Su, H., Mo, K., Guibas, L.J.: PointNet: deep learning on point sets for 3D classification and segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2017)
77. Qi, C.R., Yi, L., Su, H., Guibas, L.J.: PointNet++: deep hierarchical feature learning on point sets in a metric space. In: NIPS 2017 (2017)
78. Ravanbakhsh, S., Schneider, J., Poczos, B.: Deep learning with sets and point clouds (2016). arXiv:1611.04500
79. Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: towards real-time object detection with region proposal networks. In: Advances in Neural Information Processing Systems, pp. 91–99 (2015)

80. Riegler, G., Ulusoy, A.O., Geiger, A.: OctNet: learning deep 3D representations at high resolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, vol. 3 (2017)
81. Rusu, R.B., Blodow, N., Marton, Z.C., Beetz, M.: Aligning point cloud views using persistent feature histograms. In: 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 3384–3391. IEEE (2008)
82. Rusu, R.B., Blodow, N., Beetz, M.: Fast point feature histograms (FPFH) for 3D registration. In: IEEE International Conference on Robotics and Automation, ICRA'09, pp. 3212–3217. IEEE (2009)
83. Sabour, S., Frosst, N., Hinton, G.E.: Dynamic routing between capsules. In: Advances in Neural Information Processing Systems, pp. 3856–3866 (2017)
84. Savva, M., Yu, F., Su, H., Aono, M., Chen, B., Cohen-Or, D., Deng, W., Su, H., Bai, S., Bai, X., et al.: SHREC16 track: largescale 3D shape retrieval from ShapeNet Core55. In: Proceedings of the Eurographics Workshop on 3D Object Retrieval, pp. 89–98 (2016)
85. Savva, M., Kadian, A., Maksymets, O., Zhao, Y., Wijmans, E., Jain, B., Straub, J., Liu, J., Koltun, V., Malik, J., Parikh, D., Batra, D.: Habitat: a platform for embodied AI research (2019). arXiv:1904.01201
86. Shi, J., Dong, Y., Su, H., Stella, X.Y.: Learning non-Lambertian object intrinsics across ShapeNet categories. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 5844–5853. IEEE (2017)
87. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al.: Mastering the game of go with deep neural networks and tree search. Nature **529**(7587), 484 (2016)
88. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition (2014). arXiv:1409.1556
89. Song, S., Xiao, J.: Deep sliding shapes for a modal 3D object detection in RGB-D images. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 808–816 (2016)
90. Song, S., Yu, F., Zeng, A., Chang, A.X., Savva, M., Funkhouser, T.: Semantic scene completion from a single depth image. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1746–1754 (2017)
91. Su, H.: Deep 3D representation learning. Ph.D. thesis, Stanford University (2018)
92. Su, H., Maji, S., Kalogerakis, E., Learned-Miller, E.G.: Multi-view convolutional neural networks for 3D shape recognition. In: Proceedings of the ICCV (2015)
93. Su, H., Jampani, V., Sun, D., Maji, S., Kalogerakis, E., Yang, M.H., Kautz, J.: SPLATNet: sparse lattice networks for point cloud processing. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2530–2539 (2018)
94. Sun, J., Ovsjanikov, M., Guibas, L.: A concise and provably informative multi-scale signature based on heat diffusion. Computer Graphics Forum, vol. 28, pp. 1383–1392. Wiley Online Library, Hoboken (2009)
95. Sung, M., Su, H., Kim, V.G., Chaudhuri, S., Guibas, L.: ComplementMe: weakly-supervised component suggestions for 3D modeling. ACM Trans. Graph. (TOG) **36**(6), 226 (2017)
96. Tatarchenko, M., Dosovitskiy, A., Brox, T.: Octree generating networks: efficient convolutional architectures for high-resolution 3D outputs (2017). arXiv:1703.09438
97. Tatarchenko, M., Park, J., Koltun, V., Zhou, Q.Y.: Tangent convolutions for dense prediction in 3D. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 3887–3896 (2018)
98. Thomas, H., Qi, C.R., Deschaud, J.E., Marcotegui, B., Goulette, F., Guibas, L.J.: KPConv: flexible and deformable convolution for point clouds (2019). arXiv:1904.08889
99. Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., Abbeel, P.: Domain randomization for transferring deep neural networks from simulation to the real world. In: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 23–30. IEEE (2017)
100. Tulsiani, S., Su, H., Guibas, L.J., Efros, A.A., Malik, J.: Learning shape abstractions by assembling volumetric primitives. In: Computer Vision and Pattern Recognition (CVPR) (2017)

101. Van Den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A.W., Kavukcuoglu, K.: WaveNet: a generative model for raw audio. In: SSW, p. 125 (2016)
102. van der Maaten, L., Hinton, G.: Visualizing data using t-SNE. J. Mach. Learn. Res. **9**(Nov), 2579–2605 (2008)
103. Vinyals, O., Bengio, S., Kudlur, M.: Order matters: sequence to sequence for sets (2015). arXiv:1511.06391
104. Wang, S., Suo, S., Ma, W.C., Pokrovsky, A., Urtasun, R.: Deep parametric continuous convolutional neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2589–2597 (2018)
105. Wang, Y., Sun, Y., Liu, Z., Sarma, S.E., Bronstein, M.M., Solomon, J.M.: Dynamic graph CNN for learning on point clouds (2018). arXiv:1801.07829
106. Wang, P.S., Liu, Y., Guo, Y.X., Sun, C.Y., Tong, X.: O-CNN: octree-based convolutional neural networks for 3D shape analysis. ACM Trans. Graph. (TOG) **36**(4), 72 (2017)
107. Weinmann, M., Jutzi, B., Hinz, S., Mallet, C.: Semantic point cloud interpretation based on optimal neighborhoods, relevant features and efficient classifiers. ISPRS J. Photogramm. Remote Sens. **105**, 286–304 (2015)
108. Wu, Y., Schuster, M., Chen, Z., Le, Q.V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al.: Google's neural machine translation system: bridging the gap between human and machine translation (2016). arXiv:1609.08144
109. Wu, Z., Shou, R., Wang, Y., Liu, X.: Interactive shape co-segmentation via label propagation. Comput. Graph. **38**, 248–254 (2014)
110. Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., Xiao, J.: 3D ShapeNets: a deep representation for volumetric shapes. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1912–1920 (2015)
111. Xie, S., Liu, S., Chen, Z., Tu, Z.: Attentional ShapeContextNet for point cloud recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4606–4615 (2018)
112. Xu, D., Anguelov, D., Jain, A.: PointFusion: deep sensor fusion for 3D bounding box estimation (2017). arXiv:1711.10871
113. Xu, Y., Fan, T., Xu, M., Zeng, L., Qiao, Y.: SpiderCNN: deep learning on point sets with parameterized convolutional filters. In: Proceedings of the European Conference on Computer Vision (ECCV), pp. 87–102 (2018)
114. Yi, L., Kim, V.G., Ceylan, D., Shen, I.C., Yan, M., Su, H., Lu, C., Huang, Q., Sheffer, A., Guibas, L.: A scalable active framework for region annotation in 3D shape collections. In: SIGGRAPH Asia (2016)
115. Yi, L., Su, H., Guo, X., Guibas, L.J.: SyncSpecCNN: synchronized spectral CNN for 3D shape segmentation. In: CVPR, pp. 6584–6592 (2017)
116. Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R.R., Smola, A.J.: Deep sets. In: Advances in Neural Information Processing Systems, pp. 3391–3401 (2017)
117. Zhou, Y., Tuzel, O.: VoxelNet: end-to-end learning for point cloud based 3D object detection (2017). arXiv:1711.06396