POINT CLOUD CLASSIFICATION USING DEEP NEURAL NETWORKS

Federica Di Lauro
829470
f.dilauro2@campus.unimib.it

Andrea Premate
829777
a.premate@campus.unimib.it

Lidia Lucrezia Tonelli
813114
l.tonelli@campus.unimib.it

January 30, 2022

ABSTRACT

Point clouds are a particular type of data, made of an unordered list of 3D points in space. Feeding a deep neural network with this type of input, for tasks such as classification, segmentation and so on, is not a trivial task because there is no intrinsic geometric structure in the data, but the geometric structure of the point cloud is fundamental to extract information. In this in-depth study we analyze multiple approaches that have been proposed in literature. We limit our study to the classification task, since it's the task that have been first treated in literature and the neural network developed for it are used as backbones for more difficult tasks.

1 Introduction

3D data acquired by sensors such as LiDARs and RGB-D (RGB point and depth value) cameras, complemented with 2D images, can make machines able to understand the surrounding environment; these data have rich geometric, shape and scale information and therefore have several areas of application, including autonomous driving, robotics, remote sensing and medical treatment [1].

Point cloud understanding includes many tasks, some of which are fundamental tasks in images, such as **classification**, that aims to assign to each object in the scene a category or class, **segmentation**, that is the task of labeling each individual point (it can be semantic or instance), and **object detection**, a recent research topic which objective is to locate all the objects in a given scene. Other tasks in point cloud are tracking (estimating the location of an instance in subsequent frames), flow estimation (estimating the flow among sequences of point clouds), matching and registration (finding the relationship between point clouds of the same scene collected in different ways), augmentation and completion (improving the quality of raw point clouds), and reconstruction.

In this survey some of the main existent approaches to point cloud classification will be described.

Point Cloud A point cloud is a set of data points in space that may represent a 3D shape or object; each point is identified by a set of Cartesian coordinates (x, y, z). 3D scanners or photogrammetry software produce sets of data by measuring many points on the surfaces of objects in their surrounding environment.

3D data can have different formats: depth images, point clouds, meshes and volumetric grids; point clouds representation is a very used format because it preserves the geometric information without discretization. In contrast to image data, point clouds don't directly contain intrinsic spatial structure because they're composed by a set of 3D coordinates and they're not arranged in an array like images.

Deep learning techniques have recently been used to face many tasks in computer vision, especially image tasks; because of the difference between an image task and a 3D point cloud task, in order to apply deep learning to this type of data there are three main problems to be solved: find a dense representation from a sparse point cloud, build a network satisfying size-variance and permutation-invariance restrictions, process large volumes of data in lower time and computational resources [2].

Classification task In this survey we will describe the task of classification on point clouds, since it's the first task to have been treated in the literature. Classification on point clouds is known as 3D shape classification and it's similar to image classification: methods usually learn the embedding of each point in the point cloud, then extract a global shape embedding for the whole cloud with an aggregation encoder; the global embedding is passed through fully connected layers to obtain the expected class. Classification models can be divided in projection-based methods and point-based methods, that will be illustrated in the next sections. Some important classification models in chronological order are shown in figure 1 ([1]).

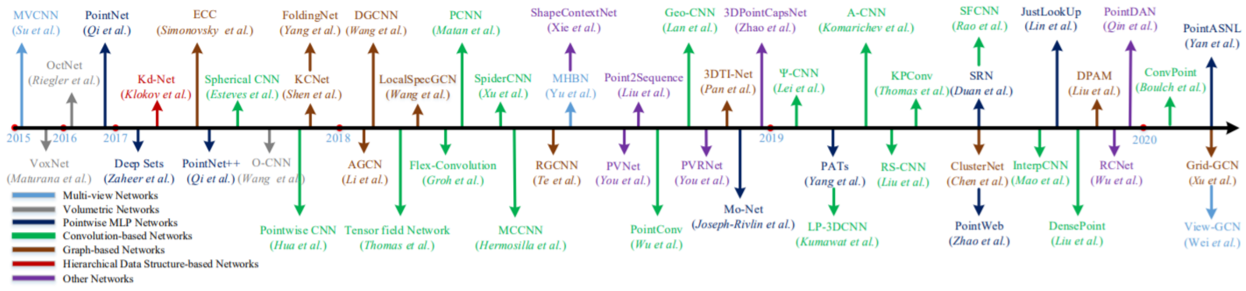

Figure 1: Relevant deep learning models for point cloud classification [1].

1.1 Datasets

Well-designed datasets are needed to evaluate the performances of deep learning algorithms on 3D point clouds applications and help defining new tasks and research topics on the subject. 3D datasets are usually small because their capture requires much more effort than 2D images, and efficiently providing dense annotations in 3D is non-trivial [3].

For the task of classification, datasets can be synthetic or real-world: the former are complete and don't have occlusions and background, the latter are occluded at different levels and some objects are contaminated with background noise.

1.1.1 Synthetic datasets

ModelNet40 [4] ModelNet is a large-scale object dataset of 3D computer graphics CAD models, constructed initially to train a 3D deep learning model called 3D ShapeNets. ModelNet is constructed from downloaded 3D CAD models labelled first with Amazon Mechanical Turk and then manually checked. It contains 151,128 3D CAD models belonging to 660 unique object categories.

ModelNet project provides three benchmarks: ModelNet10, ModelNet40 and Aligned40 (where the numbers stand for the number of categories).

This dataset, along with ShapeNet, is often used to evaluate the capacity of backbones before applying it to more complicated tasks. Since ModelNet40 is composed of CAD models and not point clouds, to use the points coordinates as input in the developed networks, the CAD models are sampled to reproduce point clouds. For example, to evaluate classification with PointNet, described later in section 2.2.1, 1024 points are sampled uniformly from the mesh faces of the CAD models, an example is shown in image 2, while figure 3 shows examples of classes and models present in ModelNet40.

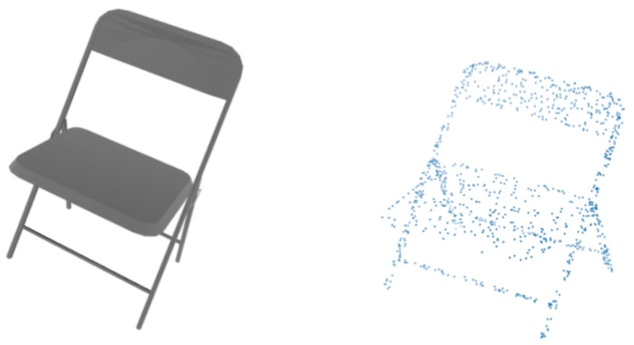

Figure 2: Left: CAD model from ModelNet40. Right: point cloud sampled randomly from the CAD model, 1024 points used.

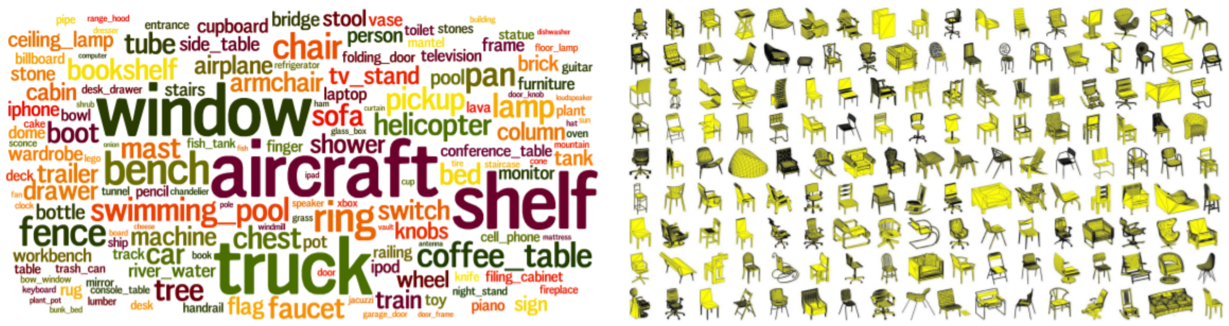

Figure 3: ModelNet40. Left: word cloud visualization of the ModelNet dataset. Right: examples of 3D chair models [4].

ShapeNet [5] ShapeNet is a repository of shapes represented by 3D CAD models for object of the everyday world, therefore it is a synthetic dataset of point clouds. It is richly-annotation and contains models spanning a multitude of semantic categories: there are 55 categories and they are organized in a hierarchical way according to WordNet synets ("synonym sets"). ShapeNet has a rich set of annotation for each shape and correspondences between them, the annotations are geometric attributes such as orientation vectors, parts and keypoints, shape symmetries and scale of object in real world units.

1.1.2 Real-world datasets

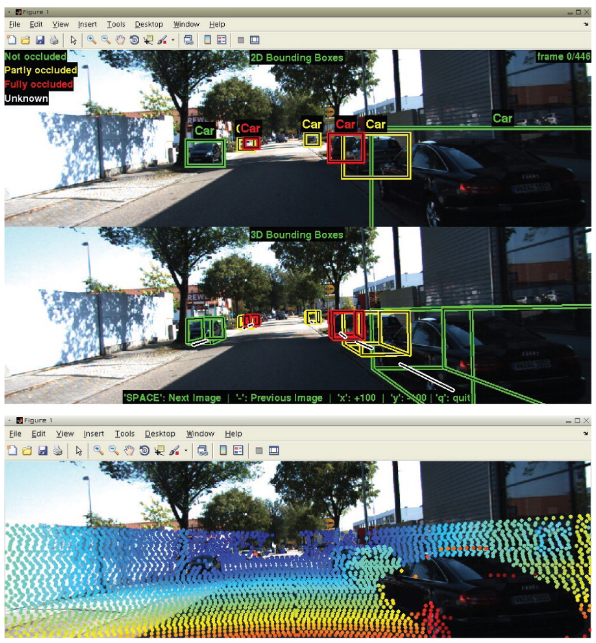

Figure 4: Images with 2D and 3D bounding boxes and labels (top), Velodyne point clouds (bottom) and their projections onto the image plane [6].

Real world datasets are acquired by using sensors such as LiDARs and RGB-D cameras. This implies the presence of noise because real-world sensors are imperfect, and acquiring data from the real world is a difficult task. Both indoor and outdoor dataset can be found.

An example of indoor dataset is S3DIS [7], a large-scale dataset composed of colored 3D scans, that are points with 3D coordinated and RGB color values, of indoor areas of large building with various architectural styles.

Outdoor datasets can vary on the type of scene acquired, but most datasets are taken in urban environments, since this type of data is particularly interesting in the context of autonomous driving.

The most notable example of outdoor dataset acquired with autonomous driving in mind is KITTI [6]: data is captured from a VW station wagon, an autonomous driving platform with two high-resolution color and grey cameras, a Velodyne laser scanner and a GPS localization system. Scenarios include real-world traffic on freeways in rural areas or inner-city scenes with many static and dynamic objects. For each sequence data include raw data, object annotations in form of 3D bounding box tracklets and a calibration file. For each dynamic object within the reference camera's field of view, the 3D bounding boxes classify objects in the classes 'Car', 'Van', 'Truck', 'Pedestrian', 'Person', 'Cyclist', 'Tram' and 'Misc'; each object has a class and its 3D size. For each frame it's available the object's translation and rotation in 3D in the Velodyne coordinates, along with the level of occlusion and truncation. An example of a scene recorded from this dataset is shown in figure 4.

1.2 Metrics

Metrics are needed in order to compare performances of different algorithms; different evaluation metrics have been proposed for every tasks in point cloud understanding, so it is important to choose the appropriate ones ([2], [1]).

For the task of **3D shape classification** the most commonly used metrics are the *Overall Accuracy (O.A.)* and the *Mean Class Accuracy (mACC)*. The O.A. is defined as the accuracy on the entire dataset and indicates how many predictions are correct over all predictions; the O.A. is defined in 1.

$$O.A. = \frac{TP + TN}{|dataset|} = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

mACC is the mean of the accuracy of every class, defined in 2.

$$mACC = \frac{1}{C}\sum_{c=1}^{C} Accuracy_c \tag{2}$$

where C is the number of classes in the dataset.

2 Classification Methods

2.1 Projection based

Projection-based methods project the unstructured 3D point clouds into a specific presupposed modality (e.g. multi-view, voxels, pillars..), and extract features from the target format, which allows them to benefit from the previous research findings in the corresponding direction. In particular we analyze the multi-view representation approach, which can take advantage of the vast literature on computer vision, and the volumetric representation approach.

While intuitively it seems logical to build 3D shape classifiers directly from 3D models, in this section it is shown how an approach based on 2D images can actually dramatically outperform classifiers built directly on the 3D representations. In particular, a convolutional neural network trained on a fixed set of rendered views of a 3D shape and only provided with a single view at test time increases category recognition accuracy by a remarkable 8% (77% $\rightarrow$ 85%) over the best models trained on 3D representations (in 2015). With more views provided at test time, its performance further increases. One of the main reasons for this performance increment is that by using a voxel-based representation the resolution needs to be significantly reduced in order to face the cubic spacial resolution. For example, 3D ShapeNets [4] use a coarse representation of shape, a 30×30×30 grid of binary voxels. In contrast, a single projection of the 3D model of the same input size corresponds to an image of 164×164 pixels, or slightly smaller if multiple projections are used. In addiction to this, must be taken into account that using 2D images allows to benefit from massive image databases, pre-existing CNN architectures which can be fine tuned and a wide literature on image descriptors.

2.1.1 Multi-View

This whole section is based on [8] except for some explicitly specified details of multi-view CNN architecture, included in [9].

As introduced before, the simple strategy of classifying views independently works remarkably well, but the focus of this section is to explain new ideas for how to “compile” the information in multiple 2D views of an object into a compact object descriptor using a new architecture called multi-view CNN. The multi-view CNN learns to combine the views instead of averaging, and thus can use the more informative views of the object for prediction while ignoring others. The multi-view approach, in the image classification context, is related to “jittering” (or data augmentation) as the different views can be seen as jittered copies. The method that will be described in fact can be also applied to jittered copies and can outperforms standard CNNs trained with the same jittered copies (for example on the sketch recognition benchmark [10]).

Let’s start from the data: the view-based representations starts from multiple views of a 3D shape, generated by a rendering engine. As said before, we could simply average the descriptors obtained from these views (that are all considered equally important) and obtain good results. Another approach is to concatenate all the different views and obtain a single big compact descriptor, but in this case we need to always respect the same point of views order and applying this order is not always possible, because the pose of the objects must be the expected one or correctly deduced. The approach we'll describe, instead, learns to combine information from multiple views using a unified CNN architecture that includes a view-pooling layer.

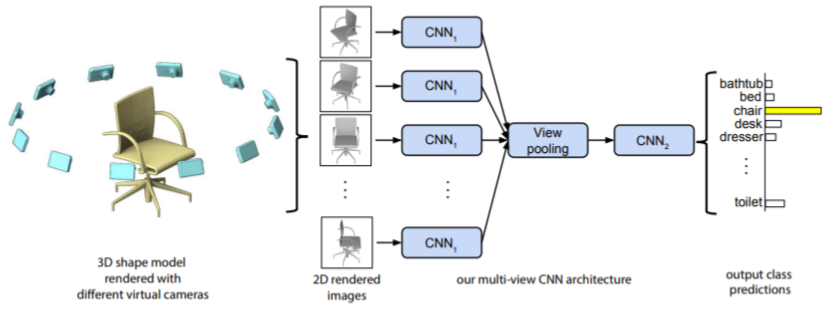

Figure 5: Multi-view CNN for 3D shape recognition (illustrated using the 1st camera setup). At test time a 3D shape is rendered from 12 different views and these are passed thorough CNN1 to extract view based features. These are then pooled across views and passed through CNN2 to obtain a compact shape descriptor [8].

Input In order to generate the different views of a 3D object, given its point cloud representation, we essentially need to preprocess the data with two steps: we connect the points and obtain edges forming faces and in order to generate rendered views of polygon meshes we need to use a reflection model (in the described approach the Phong reflection model [11] is used). Next we can apply the perspective projection and uniformly scale shapes to fit into the viewing volume. Two different cameras setup were experimented:

- **Camera setup 1:** it is assumed that the input shapes are upright oriented, as the most online database are. 12 cameras, each every 30 degrees, elevated 30 degrees from the ground plane point toward the centroid of the mesh. See the left side of figure 5.
- **Camera setup 2:** the previous assumption about the upright orientation is eliminated so we need more views to ensure a good representation. 20 cameras pointing toward the centroid of the mesh are placed at the 20 vertices of an icosahedron enclosing the shape. Then, for each camera is taken a view using 0, 90, 180, 270 degrees rotation along the axis passing through the camera and the object centroid. This camera setup therefore has a total of 80 views.

Architecture

Naive approach As described in the same paper presenting MVCNN [8], a simple approach is to fine-tune a pre-trained CNN using the different views, obtaining a descriptor for each view. Then we train one-vs-rest linear SVMs (each view is treated as a separate training sample) to classify shapes using their image features. At test time, we simply sum up the SVM decision values over all 12 views and return the class with the highest sum. Alternative approaches like averaging image descriptors lead to lower accuracy.

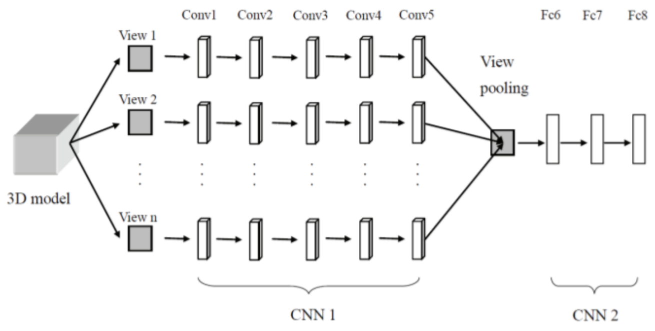

Figure 6: Architecture of MVCNN model [9].

MVCNN approach Let's describe now an approach that **learns to aggregate views: MVCNN** [9]. The input of the network are the different views of the 3D object. Each of these views is passed through the first part of the network

Table 1: Classification results. FV is another simpler approach described in the same paper that describes MVCNNs [8] based on Fisher Vectors. f.t. = fine tuning, metric = low-rank Mahalanobis metric learning

Method	Training Config. Pre-train	Training Config. Fine-tune	Training Config. # Views	Test Config. # Views	Classification (O.A.)
(1) FV	-	ModelNet40	12	1	78.8 %
(2) FV, 12×	-	ModelNet40	12	12	84.8 %
(3) CNN	ImageNet1K	-	-	1	83.0 %
(4) CNN, f.t.	ImageNet1K	ModelNet40	12	1	85.1 %
(5) CNN, 12×	ImageNet1K	-	-	12	87.5 %
(6) CNN, f.t., 12×	ImageNet1K	ModelNet40	12	12	88.6 %
(7) MVCNN, 12×	ImageNet1K	-	-	12	88.1 %
(8) MVCNN, f.t., 12×	ImageNet1K	ModelNet40	12	12	89.9 %
(9) MVCNN, f.t. + metric, 12×	ImageNet1K	ModelNet40	12	12	89.5 %
(10) MVCNN, 80×	ImageNet1K	-	80	80	84.3 %
(11) MVCNN, f.t., 80×	ImageNet1K	ModelNet40	80	80	**90.1 %**
(12) MVCNN, f.t. + metric, 80×	ImageNet1K	ModelNet40	80	80	**90.1 %**

(CNN1) that is formed by 5 convolutional layers. Consequently the quantity of branches is equal to the quantity of views. The feature mapped in output from this part are then aggregated at a view-pooling layer into a single feature map. At the end of the architecture (CNN2) we have 3 fully connected layers. See figure 6 to visualize the structure.

All branches in the first part of the network share the same parameters in CNN1. The view-pooling layer uses element-wise max pooling strategy to combine the discriminative information of multiple views and increase the computational efficiency. The view-pooling method is similar to traditional max-pooling operation. An alternative is element-wise mean operation, but it is not as effective. The view-pooling layer can be placed anywhere in the network but experiments show that setting it after the 5th convolutional layer is the best choice.

An MVCNN can also be used as a simple 2D image classifier instead of CNN. If we use data augmentation and feed the MVCNN with all the different jittered copies and the original image, we can obtain better results than using a CNN (experiments are reported in [8] on the sketch recognition benchmark [10]).

Results The dataset used is ModelNet40, described in section 1.1.1. Table 1 compares different settings of MVCNNs and the naive approach described before, based on fisher vectors or based on simply CNNs.

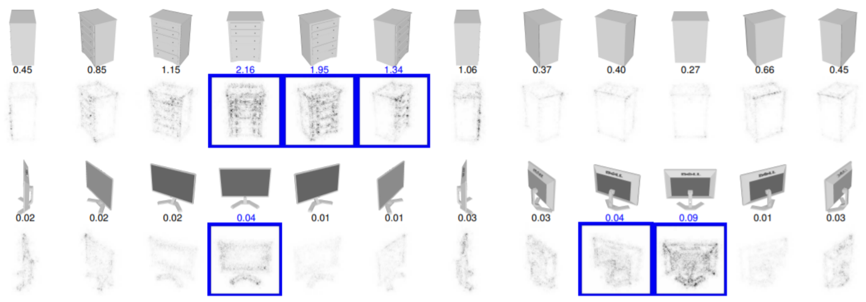

Figure 7: Top three views with the highest saliency are highlighted in blue and the relative magnitudes of gradient energy for each view is shown on top. The saliency maps are computed by back-propagating the gradients of the class score onto the image via the view-pooling layer. Notice that the handles of the dresser and of the desk are the most discriminative features [8]. (Figures are enhanced for visibility.)

Saliency map among views Since this approach learns to aggregate views in a single network it is possible to trace back to the influence of the different views on the MVCNN output score F_c for its ground truth class c. For each 3D shape S and its relative views $\{I_1, I_2, ...I_K\}$ we can compute w of the following equation using backpropagation

$$[w_1, w_2, ...w_K] = \left[\frac{\partial F_c}{\partial I_1}\Big|_S, \frac{\partial F_c}{\partial I_2}\Big|_S, ... \frac{\partial F_c}{\partial I_K}\Big|_S \right] \tag{3}$$

and obtain saliency maps for individual views. Some examples are reported in figure 7.

2.1.2 Volumetric

Since a point cloud is a 3D structure, it is quite natural using a 3D model as a projection space. In this section we will present an approach that projects the cloud points into a volumetric occupancy grids and, given these grids, performs the classification task using a 3D convolutional neural network: VoxNet [12].

Input The initial input of the algorithm is a point cloud segment, usually given by the intersection of a point cloud with a bounding box and may include background clutter.

Each point (x, y, z) of the point cloud is mapped to discrete voxel coordinates (i, j, k). This mapping depends obviously on the orientation, origin and resolution of the voxel grid in space. For the orientation it is assumed that the z axis is aligned with the gravity direction, while the origin is supposed to be given as an input. How to deal with the remaining degree of freedom, the rotation around the z axis? First method: it is possible to define a canonical orientation for each object and detect this orientation automatically, but this approach is often non-trivial in practice. Second method: using data augmentation at training time; in practise what we have to do is to create n copies of each input instance, each rotated $360°/n$ around the z axis. At testing time we pool the activations of the output layer over all n copies. As mentioned, also grid resolution has influence on results: in the experiments a fixed occupancy grid $32 \times 32 \times 32$ voxels was used. We have to keep these dimensions small because the computational space increases cubically.

There are three possible ways to model occupancy grids:

- *Binary occupancy grid.* Each voxel has just two options, it can be occupied or unoccupied. Since data coming from sensors always is noisy a probabilistic approach is used to determine the voxel's binary state.
- *Density grid.* Identical to the previous one, except for the fact that each voxel is assumed to have a continuous, and not binary, density.
- *Hit grid.* Differently from the two previous approaches, this model only consider hits, and ignore the difference between unknown and free space. Hence the probabilistic approach is different.

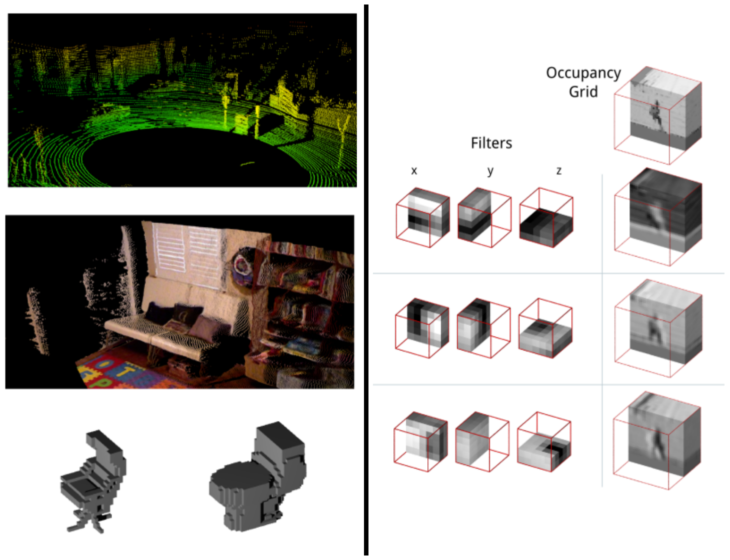

Figure 8: Left: From top to bottom, a point cloud from the Sydney Objects Dataset [13], a point cloud from NYUv2 [14], and two voxelized models from ModelNet40 [4]. Right: Cross sections of three 5 × 5 × 5 filters from the first layer of VoxNet in the Sydney Objects Database, with corresponding feature map on the right [12].

Architecture The input layer accepts a grid of $I \times J \times K$ voxels, as described before. In the performed experiments $I = J = K = 32$. Voxel entries are supposed to be normalized in the range $(-1, 1)$.

After the input layer we have convolutional layers; since we have a 3D domain, also the convolutional filters are composed by three dimensions, as shown in figure 8 on the right. The notation that will be used to describe these layers is $C(f, d, s)$, where f is the number of filters, d is the spatial dimension and s is the stride. The output is passed through a leaky ReLU with parameter 0.1 $(a(x) = max(0.1 * x, x))$.

Next we have pooling layers $P(m)$, used to downsample the input volume. We divided our input in blocks of $m \times m \times m$ and then take their maximum (we have a max-pooling operation).

At the end we have fully connected layers $FC(n)$, where n is the number of output neurons. The activation functions used are ReLUs and in the final layer it's used a softmax activation function.

An optimization of the hyperparameters leads to obtain the following network scheme: $C(32, 5, 2) - C(32, 3, 1) - P(2) - FC(128) - FC(K)$, where K is the number of classes (see figure 10). In total the model has 921736 parameters, much less than a typical CNN for image classification. The authors of the paper think this is because point cloud classification is a simpler task, as many factors that images classifiers have to take into account (perspective, illumination, ...) are not present. Training details: SGD with momentum, L2 regularization, dropout, momentum parameter 0.9, initial LR=0.01 or 0.001 depending on the dataset.

It is also possible to use a multi-resolution approach. In this case we use r networks as described before, each of them corresponding to a resolution input grid. Then, at the fully connected layers, we merge information coming from the different networks and obtain a single descriptor.

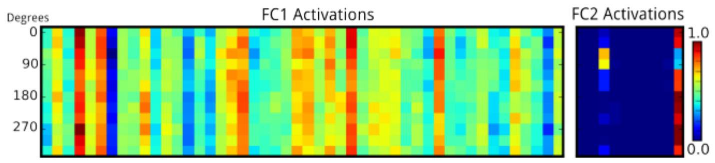

Figure 9: Neuron activations for the two fully connected layers of VoxNet when using the point cloud from figure 10 (right) as input in 12 different orientations [12].

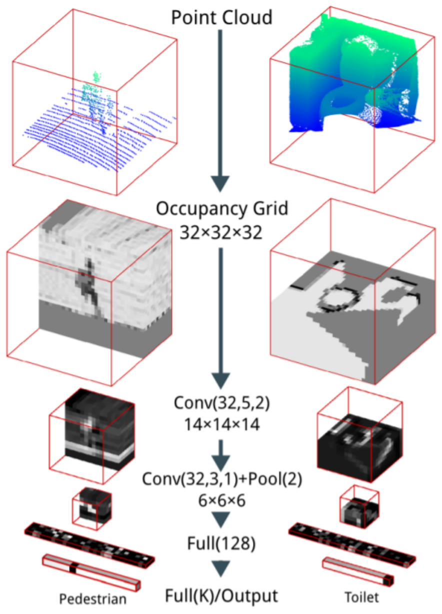

Figure 10: The VoxNet Architecture. We show inputs, example feature maps, and predicted outputs for two instances from our experiments. The point cloud on the left is from LiDAR and is part of the Sydney Urban Objects dataset [13]. The point cloud on the right is from RGB-D and is part of NYUv2 [14] [12].

Results VoxNet was evaluated in three different domains: LiDAR point cloud, RGB-D point clouds and CAD models as figure 8 shows. In figure 11 are reported some results on different datasets and comparison to other approaches.

TABLE I
EFFECTS OF ROTATION AUGMENTATION AND VOTING

Training Augm.	Test Voting	Sydney F1	ModelNet40 Acc
Yes	Yes	**0.72**	**0.83**
Yes	No	0.71	0.82
No	Yes	0.69	0.69
No	No	0.69	0.61

TABLE II
EFFECT OF OCCUPANCY GRIDS

Occupancy	Sydney F1	NYUv2 Acc
Density grid	**0.72**	**0.71**
Binary grid	0.71	0.69
Hit grid	0.70	0.70

TABLE III
COMPARISON WITH OTHER METHODS IN SYDNEY OBJECT DATASET

Method	Avg F1
UFL+SVM[21]	0.67
GFH+SVM[37]	0.71
Multi Resolution VoxNet	**0.73**

TABLE IV
COMPARISONS WITH SHAPENET IN MODELNET (AVG ACC)

Dataset	ShapeNet	VoxNet
ModelNet10	0.84	**0.92**
ModelNet40	0.77	**0.83**

TABLE V
COMPARISON WITH SHAPENET IN NYUV2 (AVG ACC)

Dataset	ShapeNet	VoxNet	VoxNet Hit
NYU	0.58	**0.71**	0.70
ModelNet10→NYU	**0.44**	0.34	0.25

Figure 11: Performance results on different datasets, and comparison to other approaches: UFL+SVM combines unsupervised Deep Learning with SVM; GFH+SVM designs a rotationally invariant descriptor and classifies it with a nonlinear SVM. [12]

A qualitative result we can test is the rotational invariance. As explained before, the network was trained using data augmentation (pre-established rotation around z axis). In order to verify if there is an effective rotational invariance we can analyze the activation of the two fully connected layers, given as input the same point cloud in different orientations. As we can see in figure 9, 12 different rotation are analyzed and the corresponding neuron activations are very similar. This shows that there is an approximate rotational invariance.

2.2 Point Based

Point-based methods learn features directly from the points and not from their spatial arrangement, as in projection based methods. These methods differ in the architecture and on the basis of that they can be divided in pointwise MLP, convolution-based, graph-based and hierarchical data structure-based. In this survey we will present MLP, convolution-based and graph-based methods.

2.2.1 Multi-Layer Perceptron

In pointwise MLP methods, features from every point of a point cloud are extracted with independent Multi-Layer Perceptrons and then aggregated with a symmetric function, such as max pooling, in a unique vector that represents the whole point cloud; the process is illustrated in figure 12 ([1]).

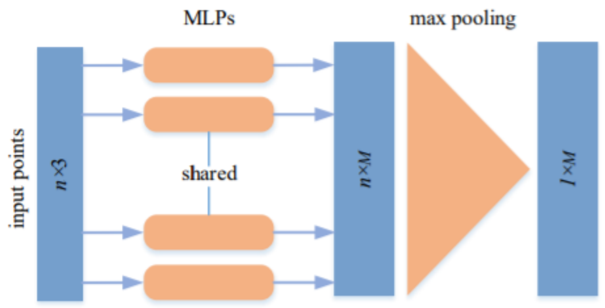

Figure 12: Architecture of PointNet [1].

PointNet A famous architecture that uses MLP and directly consumes point clouds, without transforming them in other structures, is PointNet [15], one of the early attempts to design a deep network for consumption of unordered 3D point sets; this model is used for the task of classification as well as segmentation, in fact PointNet takes as input point clouds and outputs their class labels for the entire cloud (classification) or per point (segmentation). The transformations of data made by projection-based models render the data unnecessarily voluminous while introducing quantization artifacts, point clouds directly used, instead, are easier to learn from because of their simple and unified structures. Points in a point cloud have not an intrinsic order, but the class that they represent must be invariant to permutations, so they need at least a symmetrization to be fed into the networks.

A point cloud is a set of 3D points where each point is a three-dimensional vector, and can have other features such as color channels, that we don't consider. For the classification task the input point cloud is sampled from a shape or segmented from a scene, in this case the network outputs a score for each of the k classes considered. For segmentation the input is a single object for region segmentation or a sub-volume of a scene, for this task the network outputs a score for each point and for each class.

A subset of points in an Euclidean space has three main properties: it is **unordered**, so the network that consumes n points has to be invariant to $n!$ permutations of them; the points **interact** because they are immersed in a metric space and it has to be considered the distance between them and their neighborhood; the subset is **invariant under transformations** because rotating and translating points all together does not change the class the object belongs to.

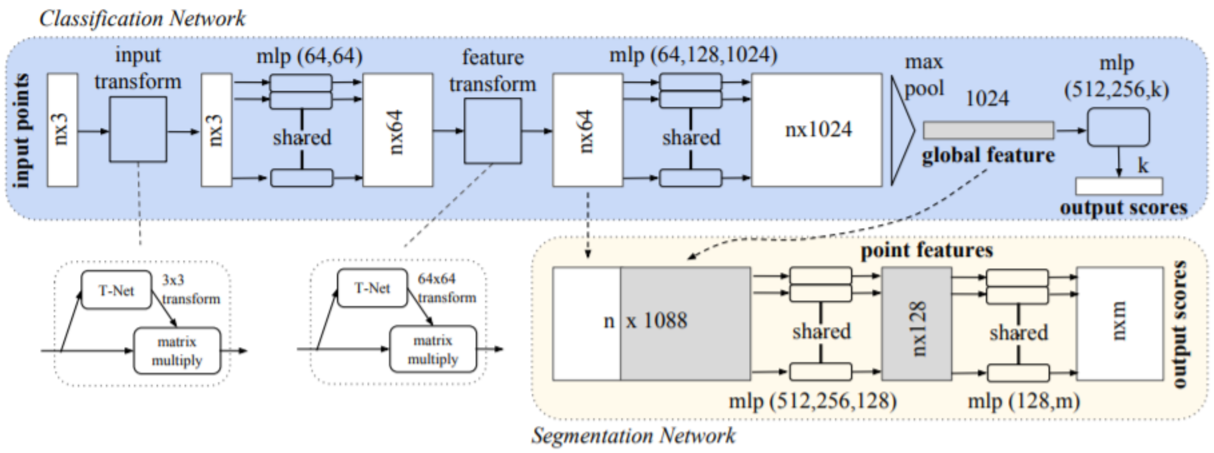

Figure 13: Architectures of PointNet for classification and segmentation tasks [15].

PointNet Architecture The input of the classification network of PointNet is a set of n points, the network then applies a transformation of this set in many layers, as can be seen in figure 13, and then aggregates features with max pooling; the global feature tensor is then passed through another MLP that produces k scores, one for every class.

Joint Alignment Network To achieve invariance to rigid geometric transformations, under which the predicted label for the point cloud must be preserved, a simple solution is to align all input set to a canonical space; instead in PointNet the T-net showed in figure 13 is used: this mini-network is composed by point independent feature extractions, max pooling and fully connected layers. The T-net predicts an affine transformation matrix and applies it to the input points; another transformation matrix is applied later in the network, to the features space that has many more dimensions than the input space. The affine transformation matrices predicted by the T-net perform a *pose normalization* to the point cloud, this also reduces the extent to which data augmentation is needed, because in point clouds the objects can take an infinite number of different poses. In order to improve optimization in such a big space, the matrix is constrained to be orthogonal by a regularization term added to the softmax training loss.

Symmetric Function for Unordered Input As previously said, the model has to be invariant to all points permutation, to achieve this objective PointNet uses a symmetric function that takes n points and produces a new order-invariant vector. The idea on which is based the symmetric module of PointNet is that a generic function defined on an unordered set of points can be approximated by a function applied on the transformed points, shown in 4.

$$f(\{x_1, \dots, x_n\}) \approx g(h(x_1), \dots, h(x_n)) \tag{4}$$

where $f : 2^{\mathbb{R}^N} \rightarrow \mathbb{R}$, $h : \mathbb{R}^N \rightarrow \mathbb{R}^K$ and $g : \overbrace{\mathbb{R}^K \times \cdots \times \mathbb{R}^K}^{n \text{ times}} \rightarrow \mathbb{R}$ is a symmetric function. In the network h is a multi-layer perceptron and g a composition of a single variable function and a max pooling function. Using a collection of h the approximated f, $[f_1, \dots, f_K]$, is produced and represents a global signature of the point set, invariant to points order. This method has a universal approximation ability proved with a theorem in [15], that means that PointNet as a neural network is capable of approximate continuous set functions (as it is f); in the worst case, with this method the network will produce a volumetric representation of point clouds by partitioning the space in equal-size voxel, but in practice it learns a better strategy.

Theoretical Analysis In [15] the robustness of the network with respect to small perturbation of noise in points is proved: thanks to the architecture of the network and the chosen functions h and g defined above, for every set of points S there exist two subsets $\mathcal{C}_S, \mathcal{N}_S \subseteq S$, such that for every set T, with $\mathcal{C}_S \subseteq T \subseteq \mathcal{N}_S$, it is true that $f(S) = f(T)$ and $|\mathcal{C}_S| \leq K$, where K is the dimension of the extracted features. In practice, $\mathcal{C}_S$ totally determines the output $f(S)$, for this reason it's called the *critical point set* and its cardinality the *bottleneck dimension of* f. This result shows that the extracted features from the point cloud remain unchanged if all the points of the critical point set are preserved and also with extra noise, up to $\mathcal{N}_S$, so PointNet is robust with respect to perturbation, corruption and extra noise points. In figure 14 critical point sets and upper-bound shapes for many objects are visualized.

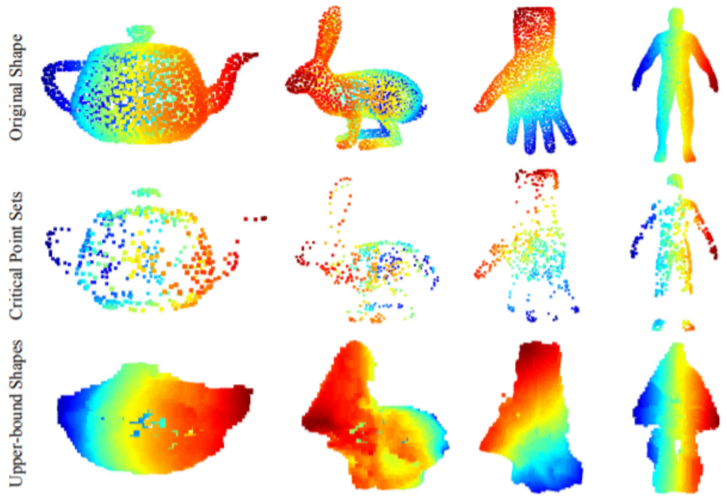

Figure 14: Critical point sets and upper-bound shapes for unseen objects [15].

For the task of classification, the extracted global features are then passed through a simple MLP for output scores. For the task of segmentation, instead, the features pass through a new part of the network that's beyond the scope of this survey.

Results Validation experiments on PointNet has been made with dataset ModelNet40, described in this survey in 1.1; PointNet has been the first MLP network tested on ModelNet40. From the mesh faces 1024 points are sampled uniformly and normalized in a unit sphere, then, during, training, these data are augmented with rotation and addition of Gaussian noise. The achieved mACC (section 1.2) is 86.2%, while the O.A. is 89.2%.

These results are achieved with the whole PointNet network, but other experiments have been conducted to demonstrate the effectiveness of input and features transformations (T-net). The performance difference with or without T-net are summarized in figure 2.

To demonstrate robustness, the network has been tested in various types of input corruption scenarios: with 50% of missing points from the input, the accuracy drops by 2.4%. At last, the analysis of space and time: PointNet has 3.5 millions parameters and has a computational cost of 440 millions FLOPs/sample (floating-point operations/sample); its space and time complexity is linear in the number of input points so it is very scalable.

PointNet++ PointNet is limited because it cannot capture local structures and fine geometric structures from the neighborhood of each point, so Qi et al. [16] proposed a hierarchical network PointNet++, inspired by the fact that CNNs take features at different scales by a stack of convolutional layers.

Table 2: Effects of input feature transformations on PointNet performances.

Transform	**Accuracy (%)**	**Description**
none	87.1	no T-net applied
input (3x3)	87.9	T-net applied only to input points (3x1)
feature (64x64)	86.9	T-net applied only to features (64x1)
feature (64x64) + reg.	87.4	T-net applied only to feature points and regularization on training loss
both	89.2	T-net applied to input points and features

PointNet learns a spatial encoding of each point and aggregates the points features in a global point cloud signature, but doing this it can't capture local structures induced by the metric of the space in which the point cloud is immersed. Using a hierarchical structure, as in CNNs, permits to progressively capture features at increasingly larger scales along a hierarchy and helps abstract local patterns to allow better generalization. PointNet++ has a *local feature learner*, that is PointNet, that abstracts sets of points or local features, the weights of these local features are shared across the partitions of the point cloud, as in the convolutional setting. The partitions are overlapping local regions, defined as a neighborhood ball in the underlying Euclidean space.

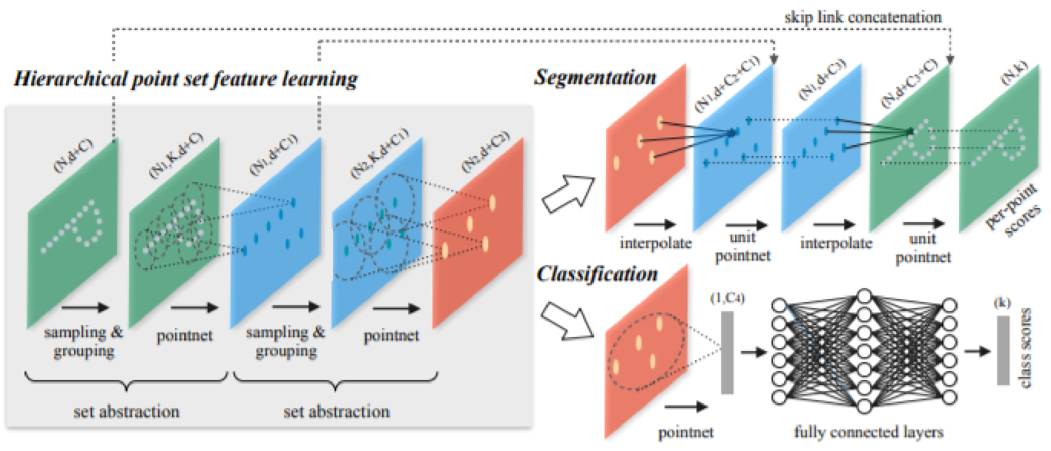

Figure 15: Architectures of PointNet++ for classification and segmentation tasks [16].

Hierarchical Point Set Feature Learning As it is shown in figure 15, PointNet++'s structure is composed by set abstraction levels, in which the set of points is abstracted and a new smaller set is produced. Every set abstraction is composed of a *Sampling*, a *Grouping* and a *PointNet* layer.

The Sampling layer selects a set of points from the input that will be the set of centroids of local regions; *iterative farthest point sampling* (FPS) is used to choose the subset of centroids from the input set: a point is chosen to be a centroid if it's the most distant point, with respect to the metric, from the centroids previously chosen than the other points in the set. The Grouping layer builds local regions by computing the neighborhood of the centroids; the neighborhood of a centroid is built with all the points that are within a radius with respect to the metric distance (while in CNN the distance between pixels is defined by Manhattan's distance). Another way to construct centroids neighborhoods is k-NN search, but the first method guarantees a fixed region scale, making local region feature more generalizable across space. The *PointNet layer* extracts feature vectors from the local regions; the input sets are the outputs of the Grouping layer, so they have different dimensions for every abstraction set, but PointNet is capable of extract fixed-length feature vectors from sets of different dimensions. Each local region is abstracted by a centroid and local feature that encodes the centroid's neighborhood, and the coordinates of points of a local region are first translated in a local frame relative to the centroid, in this way the network captures point-to-point relations in local regions.

Robust Feature Learning under Non-Uniform Sampling Density The problem of non-uniform sampling density is faced in [16], in fact features learned in dense data may not be generalized in the case of sparse data available. In the case of low density areas it is not possible to inspect closely the point cloud, so larger scale patterns are looked for. To solve this problem, some *density adaptive PointNet layers* are added to PointNet++ architecture, these layers

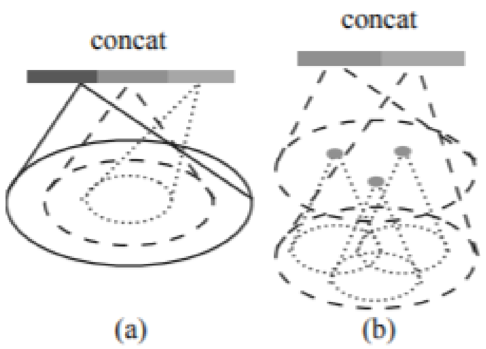

Figure 16: (a) Multi-scale grouping (MSG); (b) Multi-resolution grouping (MRG) [16].

learn to combine features from different scale regions when the density is different. So, instead of having a single scale for each abstraction level, as said above, in PointNet++ an abstraction level extracts multiple scales of local patterns and combines them intelligently according to local point density, this happens thanks to the density adaptive PointNet layers.

There are two types of density adaptive layers, shown in figure 16: *Multi-scale grouping* and *Multi-resolution grouping*. MSG layers apply grouping layers with different scales followed by an application of PointNet to extract features at each scale, then features at different scales are concatenated to form a multi-scale feature vector; MSG approach is computationally expensive since it runs local PointNets at large scale neighborhoods for every centroid. MRG layers at every level L_i build a concatenation of two vectors, the first one summarizes the features at every subregion from the previous features level in the network L_{i-1} using the set abstraction level, the second one is obtained by processing all raw points in the local region at level L_i with PointNet; when the density of a local region is low, the first vector is less reliable, but the second one is weighted higher, furthermore, when the density is high, the first vector have information of finer details.

Results PointNet++ has been evaluated on different datasets (2D objetcs, 3D objects, rigid and non-rigid objects, real 3D scenes), including ModelNet40. All the point sets are normalized to be zero mean and within a unit ball, the architecture evaluated consists in a three-level hierarchical network with three fully connected layers. The overall accuracy achieved on ModelNet40 shape classification is $91,9\%$.

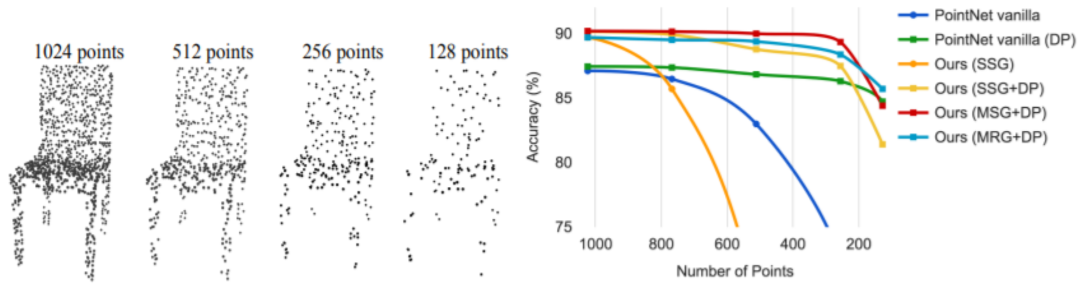

Figure 17: Left: point cloud with random point dropout. Right: accuracy w.r.t. non-uniform density [16].

PointNet++ is more robust to density variations, in fact some points are randomly dropped during training to learn an optimized strategy to combine the multi-scale features. In figure 17 it is shown the robustness of PointNet++ with MSG or MRG, and DP (random dropout), with respect to PointNet.

2.2.2 Convolutional Neural Networks

The convolution operator is defined on two functions $f(\mathbf{x})$ and $g(\mathbf{x})$, with $\mathbf{x} \in \mathbb{R}^d$:

$$(f * g)(\mathbf{x}) = \iint_{\boldsymbol{\tau} \in \mathbb{R}^d} f(\boldsymbol{\tau}) g(\mathbf{x} + \boldsymbol{\tau}) d\boldsymbol{\tau} \tag{5}$$

In images the function $g(\mathbf{x})$ is a 2D function, and because images are made up by a discrete and fixed grid of pixel the integrals can be seen as the discrete sum of product between f and g.

Such a regular structure is not intrinsic of point clouds, and thus convolution is not as easily implemented.

To tackle this problem without using an intermediate representation of the point cloud (as seen with projection based approaches) specialized convolution operators have been proposed.

In this section the PointConv operation, proposed by Wenxuan Wu et al [17], will be explored.

PointConv A point cloud is a set of 3D points (x, y, z), so the 3D convolution can be written as:

$$\text{Conv}(W, F)_{xyz} = \iiint_{(\delta_x,\delta_y,\delta_z)\in G} W(\delta_x, \delta_y, \delta_z)\, F(x+\delta_x, y+\delta_y, z+\delta_z)\, d\delta_x d\delta_y d\delta_z \tag{6}$$

where $W(\delta_x, \delta_y, \delta_z)$ is the weight function and $F(x+\delta_x, y+\delta_y, z+\delta_z)$ is the feature of a point in the local region G centered in $p = (x, y, z)$.

Point clouds, unlike images, are not uniformly sampled from the 3D space: the points $(\delta_x, \delta_y, \delta_z)$ do not have a fixed structure in the local region, so there could be subregions with more dense sampling and subregions with more sparse points, thus transforming the integral into a discrete sum is not trivial. To deal with the uneven sampling the authors introduced the inverse sparsity function $S(x, y, z)$. The PointConv operator is thus defined as:

$$\text{PointConv}(S, W, F)_{xyz} = \sum_{(\delta_x,\delta_y,\delta_z)\in G} S(\delta_x, \delta_y, \delta_z)\, W(\delta_x, \delta_y, \delta_z)\, F(x+\delta_x, y+\delta_y, z+\delta_z) \tag{7}$$

Since the weights are shared between all the points, and that PointConv is a full approximation of a convolutional layer it is both invariant to permutation of the points and to translation.

The full structure of the PointConv operator, applied on a local region with K points, can be seen in figure 18.

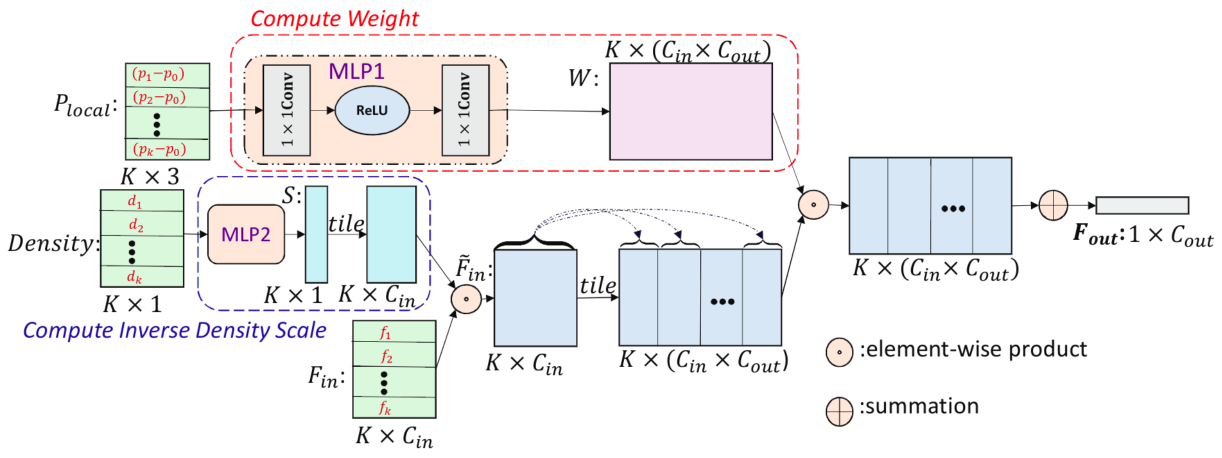

Figure 18: PointConv operator [17]

The weight function W can be approximated by MLPs, while the density function, which outputs a value indicating how much points are concentrated around the centroid region, is calculated using kernel density estimation (KDE), described in [18], and then by appling a non-linear transform using an MLP.

The inputs of the PointConv operator are:

- The subset of points $P_{local} \in \mathbb{R}^{K\times 3}$: given a point p_0 in the point cloud, the K closest points are taken and then the p_0 coordinates are subtracted to each point, obtaining the local coordinates of each point.
- The $Density \in \mathbb{R}^{K}$ estimated at each local point using KDE.
- The features $F_{in} \in \mathbb{R}^{K\times C_{in}}$, where C_{in} is the number of channels, and for each channel there are the features associated with each local point. These features can be the point coordinates (x, y, z), but also other features associated with the point such as the RGB color.

The *Compute Weight* module is made by an MLP implemented as a 1×1 convolution, followed by a non-linear ReLU activation function, followed by another 1×1. The output of this module is $W \in \mathbb{R}^{K \times (C_{in} \times C_{out})}$.

The *Compute Inverse Density Scale* module is used to calculate S: this is done by feeding the $Density$ into an MLP for a one dimensional linear transform. In this way the network can "decide" to use the density estimates.

Finally, the $F_{out} \in \mathbb{R}^{C_{out}}$ feature(s) associated with the feature(s) F_{in} is:

$$\mathbf{F}_{\text{out}} = \sum_{k=1}^{K} \sum_{c_{in}=1}^{C_{in}} S(k) \mathbf{W} (k, c_{in}) F_{in} (k, c_{in}) \tag{8}$$

Architecture of a NN using PointConv The architecture proposed in combination with PointConv is a hierarchical structure, composed by *feature encoding blocks*. In each feature encoding block there is a sampling layer, a grouping layer and a PointConv. This feature encoding blocks (apart from the PointConv operator) are very similar to the ones already seen in PointNet++ [16], see also paragraph 2.2.1. The only difference is that the neighbors of the centroid are calculated using the k-NN approach. In total the neural network is composed of 3 feature encoding blocks followed by 3 fully connected layers[1].

Efficient PointConv A noticeable problem of the original PointConv operator is the memory needed for the W function: given the batch size B, the number of points in the point cloud N, the number of local points K, the number of input channels C_{in} and the number of output channels C_{out} the size of the filter weight would be $B \times N \times K \times C_{in} \times C_{out}$.

It can be shown that PointConv equation 8 can be reduced to a matrix multiplication and a 2D convolution. The new operator can be seen in figure 19. This can reduce the memory consumption by a factor of $1/64$.

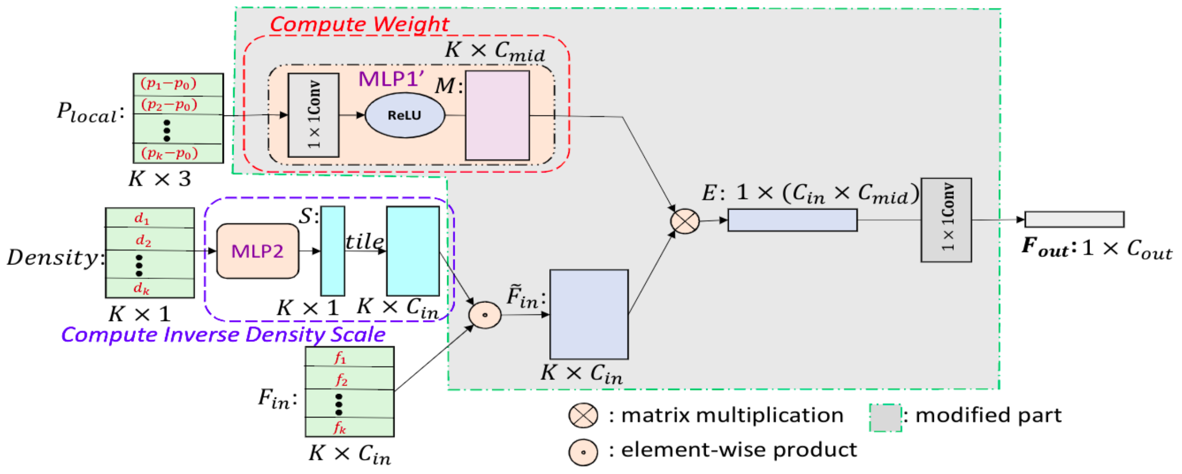

Figure 19: Efficient PointConv [17]

PointConv experiments The experiments have been performed on the ModelNet40 dataset, and have been conducted in the same way as PointNet authors did, which have become a standard to have a meaningful comparison between different networks. The overall accuracy obtained by PointConv on this dataset is 92.5

An interesting experiment that has been carried out is the classification of CIFAR-10 dataset [19], which consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. To demonstrate that PointConv is a good approximation of convolution the authors converted each point in an image into (x, y) coordinates with the associated RGB color. Two CNNs have been trained, using PointConv as the convolution operator. The accuracy results in table 3 show that the networks using PointConv have roughly the same accuracy as normal CNNs, given that the overall architecture remains the same.

[1]See the code hosted on GitHub: `https://github.com/DylanWusee/pointconv_pytorch/blob/master/model/pointconv.py`

Table 3: CIFAR-10 experiments

Network	Accuracy (%)
AlexNet [20]	89.00
VGG19 [21]	93.60
PointConv (5-layers)	89.13
PointConv (VGG19)	93.19

2.2.3 Graph inspired networks

As seen before, point clouds representations lack topological information. An approach proposed by Wang et al. [22] involves constructing graphs associated with the point cloud, and then using a novel convolution operator, *EdgeConv*.

EdgeConv The edge convolution operates on a graph constructed from local regions of the point cloud.

Let F be the feature number of each point in the point cloud. In the simplest case only the (x, y, z) coordinates of the point are considered, so $F = 3$ but other features like RGB color can be used, for example if a 3D camera was employed to acquire the point cloud. In general F is the dimensionality of the features in input to a layer.

To apply the edge convolution it must first be constructed a graph $G = (V, E)$, where $V \in \{1 \ldots n\}$ are the vertices and $E \in V \times V$ are the edges. To construct a graph which represent a local structure in the point cloud a point is taken and then its k-nearest neighbors are computed. The graph will then have the points found as vertices and the connections between the point and its neighbors as edges.

An edge-feature is defined as $e_{ij} = h_\Theta(x_i, x_j) : R^F \times R^F \to R^{F'}$, where h_Θ is a non-linear function with Θ learnable parameters. The *EdgeConv* operator is then defined by using an aggregation function $\square$ over the edge features associated with all the edges at each point. The output of EdgeConv at the vertex x_i is:

$$\mathbf{x}_i' = \square_{j:(i,j)\in\mathcal{E}} h_\Theta\left(\mathbf{x}_i, \mathbf{x}_j\right) \tag{9}$$

After each EdgeConv layer the number of points in the point cloud remains the same, the input feature dimension is F and the output feature dimension is F'.

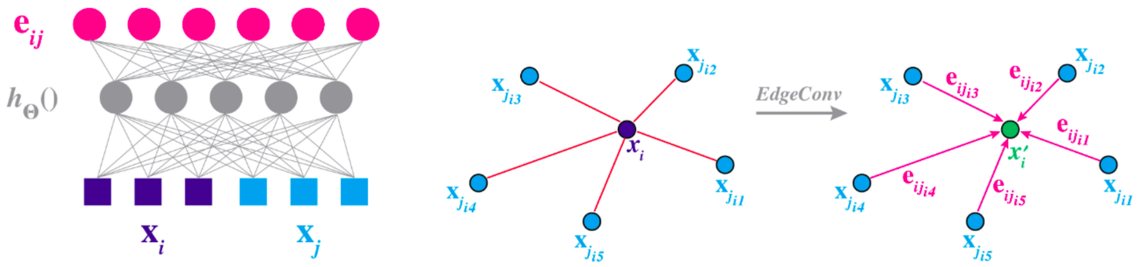

Figure 20: Left: the edge features e_{ij}, computed by a fully connected layer. Right: the edge features computed for each edge. [22]

The choice of the h_Θ and $\square$ defines the properties of the EdgeConv operator. The authors of the paper chose an asymmetric edge function:

$$h_{\mathbf{\Theta}}\left(\mathbf{x}_i, \mathbf{x}_j\right) = \bar{h}_{\mathbf{\Theta}}\left(\mathbf{x}_i, \mathbf{x}_j - \mathbf{x}_i\right) \tag{10}$$

This function takes into account both the global structure by using the coordinates of x_i and the local structure by using the distances between x_i and its neighbors. This function is easily implemented by an MLP.

As for the aggregation function $\square$ the authors chose the max function:

$$x_i' = \max_{j:(i,j)\in\mathcal{E}} e_{ij}',$$

The properties of the EdgeConv operator depend on the choice of the edge and aggregation functions.

Using the $\max$ aggregation achieves permutation invariance with respect to the order of the neighbor points x_j.

As for the translation invariance it can seen in equation 10 that the operator is partially invariant on the translation. It is easy to demonstrate that $h_\Theta(x_i - x_j)$ is translation invariant, while $h_\Theta(x_i)$ isn't, as shown in equation 11.

$$\begin{aligned} \bar{h}_{\boldsymbol{\Theta}}\left((\mathbf{x}_j + T) - (\mathbf{x}_i + T), \mathbf{x}_i + T\right) &= \\ \theta \cdot \left((\mathbf{x}_j + T) - (\mathbf{x}_i + T)\right) + \phi \cdot (\mathbf{x}_i + T) &= \\ \theta \cdot (\mathbf{x}_j - \mathbf{x}_i) + \phi \cdot (\mathbf{x}_i + T) & \end{aligned} \tag{11}$$

If only the first part is considered then EdgeConv is fully invariant to translation, however the global structure information would be lost: the classification would be based on patches of the point cloud without taking into account the global pose of these patches.

Dynamic graph update An important part explored is the dynamic graph computation. State of the art approaches in graph based network compute the graph at the beginning and then use it throughout the network. DGCNN instead recomputes the graph after each EdgeConv layer: the architecture learns how to construct the graph. The k-nearest neighbors are found by computing the pairwise distance matrix in feature space.

Architecture and results The network architecture used for the evaluation experiments is shown in figure 21. It is composed of 4 EdgeConv layers with skip connections to aggregate multi scale features, obtaining a 512-dimesional point cloud, and a max pooling layer. Then there are multiple fully connected layers. All layers include leaky ReLU as activation function and batch normalization.

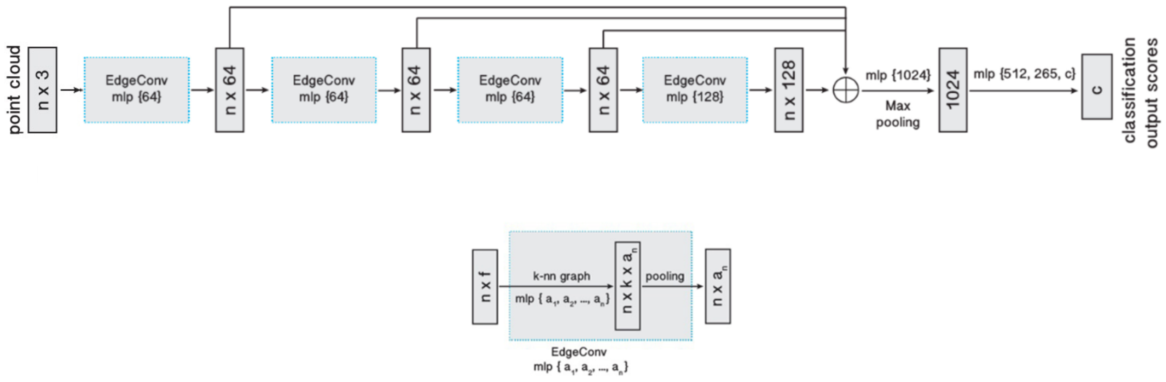

Figure 21: Top: DGCNN network architecture. Bottom: EdgeConv. [22]

The experiments have been performed on ModelNet40, and have been conducted in the same way as PointNet authors did, which have become a standard to have a meaningful comparison between different networks.

The baseline model uses only a fixed graph (no dynamic computation), and uses as edge function $h_\Theta(x_i, x_j)$. With this model an improvement of $1\,\%$ over PointNet++ is achieved.

Multiple experiments have been performed to evaluate how much each part of the model influences the accuracy results, as shown in table 4. Three improvements to the networks have been tried:

- Centralization, which is using explicitly the global information given by the vertex x_i and the distance between its neighbors, by using the edge function $h_{\boldsymbol{\Theta}}\left(\mathbf{x}_i, \mathbf{x}_j - \mathbf{x}_i\right)$.
- Dynamic graphs, which is the computation of the neighbors on the features extracted for each EdgeConv layer.
- More points, by using 2048 points instead of 1024.

It is also worth noticing that DGCNN performs faster than state of the art network such as PointNet++, while keeping the model size relatively small, as shown in table 5.

Table 4: CENT denotes centralization, DYN denotes dynamical graph recomputation and MPOINTS denotes experiments with 2,048 points

CENT	DYN	MPOINTS	MEAN CLASS ACCURACY(%)	OVERALL ACCURACY(%)
x			88.9	91.7
x	x		89.3	92.2
x	x	x	90.2	92.9

Table 5: Size and time comparison between PointNet, PointNet++ and DGCNN

	MODEL SIZE(MB)	TIME(MS)
POINTNET (BASELINE) (QI ET AL. 2017B)	9.4	6.8
POINTNET (QI ET AL. 2017B)	40	16.6
POINTNET++ (QI ET AL. 2017C)	12	163.2
DGCNN (BASELINE)	11	19.7
DGCNN	21	27.2

3 Comparison

From this survey we can see that the main points on which different networks differ is how the input is fed into the network, how each network achieves permutation and transformation invariance, and how each network exploits the spatial information of the point clouds.

Input data The main difference between the networks is what input they take to be processed.

Projection based methods don't directly use the point clouds, instead they project the point cloud into a different structure, and then feed it to the network. This makes it easier to design the network, because traditional CNNs can be used to process both images and 3D grids. The disadvantages of this approaches are the memory consumption since multiple images and voxels occupy more space than the raw point cloud, and the fact that the projection of the points can introduce artifacts in the images and 3D grids.

Point based methods directly feed the points of the point cloud in the neural network. This solves the disadvantages of projection based methods, but introduces other complications. The networks must be designed to deal with permutation of the points, transformations of the point cloud and have to specifically design methods to employ the spatial information.

Permutation invariance Permutation invariance allows the network to be unsensitive to the order of the points in the point cloud.

Projection based methods do not need an explicit way to handle the permutation of the points in the point cloud, since the points get projected either on images or voxels before being fed to the network. The methods used to project the points are invariant to the permutations of the point cloud.

Point based methods seen in this survey have different ways to achieve this property:

- PointNet and PointNet++ use a symmetric function that takes a set of points and produces a new order-invariant vector, such as a max pooling function.
- PointConv uses weights that are shared between all the points.
- DGCNN uses a symmetric aggregation function over the edge features (the authors used the max function).

Transformation invariance Transformation invariance allows the network to be robust to geometric transformations such as rotation and translation.

Projection based methods share a very similar technique:

- MVCNN uses multiple views of the same 3D object to perform the training and the classification of a point cloud. If two inputs of this network represent the same object, but rotated, the architecture will assure an approximate rotational invariance due to the optimization process. It is not intrinsically invariant to translation.

- A similar reasoning can be applied to VoxNet, with the only difference that we deal with 3D grids instead of 2D views. As explained in the Voxnet results section and in figure 9, by analyzing the neuron activations of 12 different rotation of the same object we can show that there is an approximate rotational invariance. It is not intrinsically invariant to translation.

Point based methods employ different techniques:

- PointNet and PointNet++ use T-Nets to achieve invariance to rigid transformations (both translation and rotation). T-nets predict affine transformation matrices that perform a pose normalization. A T-net predicts the matrix to be multiplied by the input points, and another T-Net predicts the matrix to be multiplied by the features.
- PointConv is a full approximation of the convolution operator, thus is invariant to translation. It is not intrinsically invariant to rotation.
- DGCNN is partially invariant to translation: it is only invariant in the part of the edge function that depends on the difference between the centroid and its neighbors, while the global information in the edge function is not translation invariant. It is not intrinsically invariant to rotation.

Spatial information Since point clouds represent 3D information in space, it can be useful to use the geometric relationship between the points.

Projection based methods project the points into fixed grids and then use standard convolution to compute the features, so they make use of the relationship between the points.

Point based methods vary depending on the network:

- PointNet doesn't use any spatial information to learn the features as it takes the whole point cloud as input.
- PointNet++ uses the feature abstraction layer to find local neighbors and then apply PointNet locally to extract a hierarchy of features.
- PointConv uses the same feature abstraction layer to find the neighbors and then applies a convolution operator.
- DGCNN first computes a local neighborhood graph and then applies a convolution-like operator on the edges of the graph.

Results comparison The only dataset on which all neural networks have been tested is ModelNet40. In table 6 the overall accuracy results and model size are shown.

It can be seen that the volumetric model has less parameters than to other networks described, but its accuracy is quite low, probably because of the resolution of the voxel grid that must be kept small because of the computational complexity increases cubically. The multi-view approach have fair accuracy results but its model size is 10 to 100 times larger than the other neural networks that achieve similar accuracy.

Point based methods performances have been achieving increasingly better accuracy results, since the first approach described by PointNet was introduced. All the point based methods described have comparable model sizes, apart from PointConv which has $\sim$10 times the parameters of PointNet++ and DGCNN, with similar accuracy results.

Table 6: Results for each network, tested on ModelNet40. The model size is taken from [1] for all models except MVCNN, for which the parameters have been extracted from the PyTorch model available on GitHub `https://github.com/jongchyisu/mvcnn_pytorch`

	MODEL SIZE (params)	ACCURACY (%)
MVCNN	128M	90.1
VoxNet	921K	83.0
PointNet	3.5 M	89.2
PointNet++	1.48 M	91.9
PointConv	11 M	92.5
DGCNN	1.84 M	92.9

4 Conclusion

In this survey we have investigated the main point cloud classification approaches: projection-based and point-based. Projection-based approaches transform the unstructured 3D point clouds into specific modality, such as multi-view, voxels or pillars, and extract features from the target format. Point-based approaches, on the other side, learn features directly from the points and not from their spatial arrangement.

As examples of projection-based approaches, we've described the multi-view and the volumetric methods; in the former the information in multiple 2D views of an object is compiled in a compact descriptor with a CNN, in the latter the cloud points are projected into volumetric occupancy grids on which a CNN performs the classification task. As examples of the multi-view method we have the *CNNs 12x*, *FV 12x* and *MVCNN*, while the volumetric approach is represented by *VoxNet*.

In the context of point-based approach, the three main methods are MLP, where features from every point are extracted with independent multi-layer perceptrons, CNN, networks that are directly fed with points and define a special type of convolution, and graph inspired networks, that exploit the topological information already present in the point cloud. As representants of these three methods we have respectively PointNet and PointNet++ as MLPs, PointConv as as CNN and EdgeConv as a graph inspired network.

The different networks described use different methods to achieve the same properties needed for treating point clouds: permutation invariance, transformation invariance and use of spatial information, a comprehensive comparison was made in section 3.

It is also worth noting that while the classification networks here described, tested on an easy, synthetic dataset might seem not really useful in real case applications, they are used as backbones for more complicated and interesting tasks, described in section 1. For example we can see from the survey by Zhang et al. [23] some neural networks that do point cloud registration are based on the methods described above: PointNetLK [24] is based on PointNet, DeepVCP [25] is based on PointNet++, DCP [26] is based on DGCNN.

References

[1] Yulan Guo, Hanyun Wang, Qingyong Hu, Hao Liu, Li Liu, and Mohammed Bennamoun. Deep learning for 3d point clouds: A survey, 2020.

[2] Haoming Lu and Humphrey Shi. Deep learning for 3d point cloud understanding: A survey, 2021.

[3] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas A. Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 2432–2443. IEEE Computer Society, 2017.

[4] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 1912–1920. IEEE Computer Society, 2015.

[5] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. Shapenet: An information-rich 3d model repository, 2015.

[6] A Geiger, P Lenz, C Stiller, and R Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013.

[7] Iro Armeni, Ozan Sener, Amir Roshan Zamir, Helen Jiang, Ioannis K. Brilakis, Martin Fischer, and Silvio Savarese. 3d semantic parsing of large-scale indoor spaces. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 1534–1543. IEEE Computer Society, 2016.

[8] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik G. Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 945–953. IEEE Computer Society, 2015.

[9] C. Li H. Zeng, Q. Wang and W. Song. Learning-based multiple pooling fusion in multi-view convolutional neural network for 3d model classification and retrieval. *Journal of Information Processing Systems*, 15(5):1179–1191, 2019.

[10] Mathias Eitz, James Hays, and Marc Alexa. How do humans sketch objects? *ACM Trans. Graph. (Proc. SIGGRAPH)*, 31(4):44:1–44:10, 2012.

[11] B. T. Phong. Illumination for computer generated pictures. In *Graphics and image processing, Commun. ACM, 18(6)*, 1975.

[12] Daniel Maturana and Sebastian Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 922–928, 2015.

[13] A. Quadros, J.P. Underwood, and B. Douillard. An occlusion-aware feature for range images. In *2012 IEEE International Conference on Robotics and Automation*, pages 4428–4435, 2012.

[14] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from rgbd images. In Andrew Fitzgibbon, Svetlana Lazebnik, Pietro Perona, Yoichi Sato, and Cordelia Schmid, editors, *Computer Vision – ECCV 2012*, pages 746–760, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[15] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation, 2017.

[16] Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space, 2017.

[17] Wenxuan Wu, Zhongang Qi, and Fuxin Li. Pointconv: Deep convolutional networks on 3d point clouds. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 9621–9630. Computer Vision Foundation / IEEE, 2019.

[18] Berwin A. Turlach. Bandwidth selection in kernel density estimation: A review. In *CORE and Institut de Statistique*, 1993.

[19] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, Canadian Institute for Advanced Research, 2009.

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[21] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[22] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph cnn for learning on point clouds. *ACM Trans. Graph.*, 38(5), 2019.

[23] Zhiyuan Zhang, Yuchao Dai, and Jiadai Sun. Deep learning based point cloud registration: an overview. *Virtual Reality & Intelligent Hardware*, 2(3):222–246, 2020. 3D Visual Processing and Reconstruction Special Issue.

[24] Yasuhiro Aoki, Hunter Goforth, Rangaprasad Arun Srivatsan, and Simon Lucey. Pointnetlk: Robust amp; efficient point cloud registration using pointnet. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7156–7165, 2019.

[25] Weixin Lu, Guowei Wan, Yao Zhou, Xiangyu Fu, Pengfei Yuan, and Shiyu Song. Deepvcp: An end-to-end deep neural network for point cloud registration. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 12–21, 2019.

[26] Yue Wang and Justin Solomon. Deep closest point: Learning representations for point cloud registration. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 3522–3531, 2019.