The International Journal of Advanced Manufacturing Technology (2023) 124:3067–3082
https://doi.org/10.1007/s00170-022-09791-z

ORIGINAL ARTICLE

Primitive shape recognition from real-life scenes using the PointNet deep neural network

Senjing Zheng[1] · **Marco Castellani**[1]

Received: 16 December 2021 / Accepted: 13 July 2022 / Published online: 2 August 2022

Abstract
In many industrial applications, it is possible to approximate the shape of mechanical parts with geometric primitives such as spheres, boxes, and cylinders. This information can be used to plan robotic grasping and manipulation procedures. The work presented in this paper investigated the use of the state-of-the-art PointNet deep neural network for primitive shape recognition in 3D scans of real-life objects. To obviate the need of collecting a large set of training models, it was decided to train PointNet using examples generated from artificial geometric models. The motivation of the study was the achievement of fully automated disassembly operations in remanufacturing applications. PointNet was chosen due to its suitability to process 3D models, and ability to recognise objects irrespective of their poses. The use of simpler shallow neural network procedures was also evaluated. Twenty-eight point cloud scenes of everyday objects selected from the popular Yale-CMU-Berkeley benchmark model set were used in the experiments. Experimental evidence showed that PointNet is able to generalise the knowledge gained on artificial shapes, to recognise shapes in ordinary objects with reasonable accuracy. However, the experiments showed some limitations in this ability of generalisation, in terms of average accuracy (78% circa) and consistency of the learning procedure. Using a feature extraction procedure, a multi-layer-perceptron architecture was able to achieve nearly 83% classification accuracy. A practical solution was proposed to improve PointNet generalisation capabilities: by training the neural network using an error-corrupted scene, its accuracy could be raised to nearly 86%, and the consistency of the learning results was visibly improved.

Keywords Primitive shape recognition · Remanufacturing · Robotic manipulation · Point cloud · Deep neural network · PointNet · Shallow neural network

1 Introduction

Reliable object manipulation procedures are a fundamental prerequisite for the robotic handling of parts in disassembly and remanufacturing. The literature on grasping and manipulation includes methods based on properties of the objects like their appearance and geometry [1, 2], or their dynamics [3]. Regardless of the method used, the shape of the target object usually needs to be estimated.

Object shape can be estimated from 2D camera images or 3D point clouds. Due to the variable appearance and state of used parts, and the loss of structural information, methods based on 2D images often fail to obtain acceptable results in remanufacturing applications [4, 5]. For this reason, 3D models are often preferred.

Thanks to the increasing availability of reliable and affordable sensors, 3D scans have nowadays become easily obtainable from the field. However, point clouds pose their own challenges due to their lack of topological structure, and the large amount of information they carry (usually millions of data points).

This study aims to investigate the ability of the PointNet deep neural network (DNN) [6] to recognise primitive shapes in point cloud models of everyday objects, after being trained on computer-generated geometric primitives. PointNet directly takes the elements of the point cloud as input, and is able to recognise objects irrespective of their position and orientation. PointNet can also be trained to segment parts and sub-assemblies from the point cloud

✉ Marco Castellani
m.castellani@bham.ac.uk

Senjing Zheng
senjing.zheng@gmail.com

[1] School of Engineering, University of Birmingham, Edgbaston, Birmingham B15 2TT, UK

scene. These features make it an ideal candidate for object recognition in a highly unstructured domain such as the disassembly and manipulation of end-of-life products. They also distinguish PointNet from standard DNNs, particularly those based on convolutional layers, since the latter require structured input data, and their internal representation of the input is generally not rotation invariant [6].

Zheng et al. [7] used CAD-generated models to train PointNet to identify complex mechanical parts for disassembly applications. Experimental evidence indicated the viability of the proposed approach, although the accuracy of the trained DNN was validated on artificial scenes created via a depth-camera simulator. This study aims to evaluate the ability of PointNet to recognise object shapes from real scans of objects, after being trained on artificial geometric models. The focus of this study is also on the abstraction of primitive shape information, rather than the recognition of detail-rich objects like car turbocharger components [7].

The research has direct application to many engineering problems beyond the disassembly and remanufacturing domain, since mechanical objects have often fairly regular shapes, which can be approximated with geometric primitives such as spheres, boxes, and cylinders (e.g. the cylindrical head of a piston, the spheres of a rolling bearing) [8, 9].

This study also aims to compare the performance of PointNet to the performance of simpler classifiers like shallow neural networks (SNNs). Provided a simple method is available to extract a structured and meaningful representation of the point cloud scenes, usually a set of features, SNNs are preferable for their comparable ease of training and low computational overheads. In this study, the performance of PointNet was compared to the performance of two popular shallow neural networks: a multi-layer perceptron (MLP) [10] and a radial basis function network (RBFN) [11].

The main difficulty in the recognition task comes from the fact that the shape of the scans is often not perfectly regular. Sensor imprecision and occlusion (partial view) further complicate the problem. In this study, the performance of PointNet was tested on real scans of common objects from the Yale-CMU-Berkeley (YCB) benchmark set [12], a popular robotics benchmark.

The use of real scans of physical objects constitutes a more realistic setting than the CAD-generated images used in the tests performed by the creators of PointNet [6], or the simulated scans used by Zheng et al. [7]. The fact that PointNet had not been evaluated on real-life point cloud model sets was first pointed out by Garcia-Garcia et al. [13], and later acknowledged by Uy et al. [14] who manually built the ScanObjectNN set. ScanObjectNN contains camera scans of physical objects grouped in categories modelled on the popular ModelNet40 benchmark set of CAD models [15]. In their study, Uy et al. [14] reported very poor classification accuracy (32.2%) when the the PointNet was trained using ModelNet40 and tested on the ScanObjectNN set.

The objects used in this study have a more regular shape than those featured in the ModelNet40 and ScanObjectNN sets. PointNet will be trained using a set of geometric primitive shapes, and then used to recognise similar shapes from real-life scenes. The obvious advantage of this arrangement is the possibility of generating an arbitrarily large model set for training the DNN, removing the need of acquiring a database of object scans.

This paper is organised as follows. Related work is discussed in Sect. 2, whilst the PointNet deep architecture is described in Sect. 3. Section 4 describes the SNN architectures, and presents the extraction scheme generating the features they use. The model sets used in the experiments are described in Sect. 5. The experimental setup and results are reported in Sect. 6, whilst the outcomes of the tests are discussed in Sect. 7. Section 8 concludes the paper.

2 Related work

Deep neural networks have gained wide popularity for 2D machine vision applications, thanks to their high accuracy and feature extraction ability [16]. In recent years, DNN-based vision technology found increasing application in the fields of manufacturing [17–19] and remanufacturing [20–25].

In detail, Yildiz and Wörgötter [24, 25] developed a screw detection and classification system based on a deep convolutional neural network, and demonstrated its accuracy in a hard disk drive disassembly case study. The creation of the training set of examples for the DNN entailed a large effort, where 20,000 sample images of 500 screw elements were collected from 50 hard disk drives. Foo et al. [21] used deep learning for screw detection in an LCD monitor disassembly application. The system used an image preprocessing procedure, an ontology reasoning module, and a Fast-RCNN network [26]. The training dataset was built combining numerous images of screws acquired via an extensive Google search, plus 356 manually acquired images. A total of 1496 bounding boxes around the screw samples had to be manually labelled in the images. Li et al. [22] used a fast region-convolution neural network to detect screws on motherboards of mobile phones for disassembly. The training procedure needed the manual acquisition of 488 images.

Brogan et al. [20] proposed a vision system based on the Tiny YOLO v2 (Tiny-You Only Look Once v2) pretrained DNN architecture, to identify screws on electrical waste for disassembly. The system achieved over 92% recognition accuracy using 900 manually collected training images. A YOLO (v4) architecture was used also by Rehnholm [23] to build the vision system for a battery package disassembly application. The training procedure required

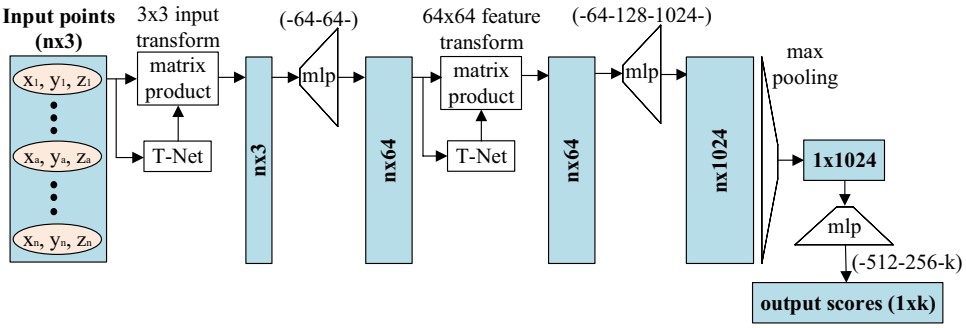

Fig. 1 Structure of the classification part of PointNet

the creation of nearly 25,000 images in total for training and validation.

In summary, although DNNs generally achieve good recognition accuracies, they require a large dataset of individually labelled images. Moreover, commonly used convolutional neural network (CNN) architectures can only process structured data such as 2D images.

With the development of 3D sensors like RADAR (radio detection and ranging), LiDAR (light detection and ranging), and RGB-D (red, green, blue, and depth channels) camera, 3D data can be easily obtained from field. Typically, the raw data is in the form of point cloud, an unordered set of data points (X, Y, Z coordinates) delineating the surfaces of the scanned object. Although depth information adds valuable context for the identification task, the lack of structure and the uneven distribution of the data points constitutes a challenge for the recognition algorithm.

In particular, the unstructured characteristic of point cloud models cannot be handled by convolutional architectures. For this reason, four main DNN approaches can be identified in the literature. Three of these methods use standard DNN architectures, often including convolutional layers, and feed these architectures with point cloud representations where the information is structured via volumetric [15, 27], multi-scene [28], or graph-/tree-based methods [29–31]. The fourth method directly processes the raw point cloud via purpose-designed DNNs like the PointNet architecture [6] used in this work. For a more detailed discussion of deep learning methods for point cloud understanding, the reader is referred to a recently published survey by Guo et al. [32].

PointNet [6] was the first DNN architecture to be able to process directly point cloud scenes. PointNet can be used to perform shape identification or segmentation, and is able to recognise objects regardless of their rotation and translation. Being able to process directly point cloud scenes, PointNet does not require computationally intensive pre-processing steps which may also cause information loss. These features immediately made PointNet very popular for the recognition of real life scenes, and spawned several similar architectures [33–35].

Zheng et al. [7] used PointNet to recognise components of two different types of turbochargers for disassembly purposes. The DNN was trained using CAD models of the automotive parts, and tested on point clouds generated using a depth camera simulator. The simulator allowed replicating various degrees of sensor imprecision and partial occlusion of the objects. The PointNet achieved classification accuracy above 90%, although its performance degraded with the addition of simulated sensor imprecision to the model test set. Zheng et al. [7] showed that the effect of sensor imprecision could be counteracted by adding comparable noise to the training data. The method has not been tested yet on real-life images, where the level and distribution of sensor error is not known.

3 The PointNet deep neural network

PointNet was proposed by Qi et al. [6] for object classification and segmentation for point cloud models. It is a DNN constituted of multiple neural layers as shown in Fig. 1, and can be divided into three key modules.

The first module is designed to map the input space to a higher-dimensional representation (embedding space), and makes the procedure invariant to rigid transformations of the object's pose. Differently from the Spatial Transformer proposed by Jaderberg et al. [36], a mini-network (T-Net) is used in PointNet [6]. The T-Net takes all the points from the point cloud as input, and predicts the affine transformation matrix that aligns the object to a canonical space before feature extraction. Additionally, another T-Net ('feature transform' in Fig. 1) is used to further align the embedding space.

The second module is the feature extraction module: it is composed of a set of MLPs and a max pooling function [6]. The MLPs are used as feature detectors that are applied to the higher-dimensional embedding space, whilst the max pooling layer is used to aggregate the feature detection result. The overall action of the first two modules is to transform the input information into a feature set. That is, it implements a symmetric function that maps the spatial information in the point cloud to the feature space, irrespective of the object pose.

The third module of PointNet is a fully connected layer that takes the feature information and generates the identification result.

In summary, when a point cloud consisting of (n) points is fed to PointNet, the coordinates of all its points are mapped into the feature space through the first and second modules of the network. The third module of the network is a standard classifier that takes the features extracted in the previous layers, and outputs the classification score for the input scene.

4 Shallow neural network architectures

Shallow neural networks (SNNs) have a longer history than DNNs. Compared to DNNs, SNNs are known to be faster to train, and are less likely to overfit the training data because they use a much smaller number of parameters (weights). Their main limitation is that they need a pre-processing step to extract the vector of input features (variables). In DNNs, feature extraction is performed by the first layers of the architecture, and is optimised by the learning procedure together with the classifier proper (the last layers of the architecture). Nonetheless, when fed with a descriptive set of features, SNNs are known to reach accuracies comparable to those obtained by DNNs [37] in point cloud classification problems.

In this study, the performance of two classical SNN models will be compared to the performance of PointNet. The first SNN is the widely used multi-layer perceptron (MLP). MLP is a popular feed-forward and versatile neural network used for classification and modelling problems [38]. The MLP is usually trained using the back-propagation learning algorithm, which was firstly proposed by Rumelhart et al. [39]. The versatility of the MLP stems from its ability to approximate any function to any desired degree of accuracy [40].

The second is the radial basis function network (RBFN). RBFN was firstly proposed by Broomhead and Lowe [11]. Like the MLP, the RBFN is a popular feed-forward neural network used for modelling and classification problems [38]. The RBFN has a strictly defined architecture, featuring one input layer, one hidden layer, and one output layer. The activation function of the hidden layer is the radial basis function. The input layer of an RBFN acts as a buffer, and broadcasts the input vector to each neuron in the hidden layer. The hidden neurons process the input vector via the activation function (radial basis function), whilst the output neurons perform a linear summation of the weighted outputs of the hidden neurons.

4.1 Numerical feature generation

A point cloud is a data structure containing an unordered list of x-y-z coordinates. Differently from PointNet, the MLP and RBFN treat the input as a vector, and are thus sensitive to the ordering of its elements. Therefore, point clouds cannot be fed directly to the MLP or RBFN. In this section, a feature generation scheme to extract numerical features from the point clouds is described.

In the tests, it is assumed that the objects are in unknown orientation. Thus, the extraction process starts with aligning the shapes with the coordinate axes. For this purpose, principal component analysis [41, 42] was used to extract the eigenvectors of the point cloud. The point cloud is then placed with its centroid in the origin, and its eigenvectors are aligned with axes of the Cartesian coordinates. Since the eigenvectors broadly correspond with the main axes of the shapes, this method was proven to align the point cloud with reasonable accuracy.

In real life, a human eye can recognise a primitive shape from its orthogonal projections onto the three planes $x = 0$, $y = 0$, and $z = 0$. Namely, a cube with its sides aligned with the Cartesian axes will create three rectangular shapes (one per plane), a sphere will generate three disks, and a cylinder will create two rectangular shapes and one disk. In summary, recognising the three 3D primitive shapes boils down to recognising two 2D shapes (rectangle and disk) in their projections. This idea is exploited as follows.

After principal component analysis alignment, the points in the cloud are projected onto the three planes $x = 0$, $y = 0$, and $z = 0$. For example, the projection of a point of coordinates $z = (x, y, z)$ onto the $z = 0$ plane is $z = (x, y, 0)$. For each 2D projection on a plane:

1. The coordinates of all the points are transformed into 2D polar-coordinates: (r, θ).
2. The plane is divided into 64 sectors (intervals of θ).
3. For each of the 64 sectors, the radius r of the most distant point from the origin is taken as representative of the interval. Namely, the representative of interval $1 \le j \le 64$ is $m_j = max(r_i)$ where $1 \le i \le N$ indicates one of the N points of the cloud.
4. The arithmetic mean and standard deviation of the 64 representatives m_j of the interval are calculated for each plane.

The features extracted from one point cloud form a 6-dimensional vector:

$$(m_x, \delta_x, m_y, \delta_y, m_z, \delta_z)$$

where m_k and δ_k $(k = x, y, z)$ are respectively the mean and standard deviation of the 64 representatives on the planes $x = 0$, $y = 0$, and $z = 0$. In a perfect point cloud without error, all the most distant points of a disk will be at the same distance from the origin; hence, $m_x = m_y = m_z$ and $\delta_x = \delta_y = \delta_z = 0$. Also, the most distant points of a rectangle will not be at the same distance from the origin, $\delta_{x,y,z} \neq 0$,

and in general $m_x \neq m_y \neq m_z$. This will hold as long as the error level is reasonable, namely that it will not completely blur the shapes of the projections, or when the model has been cleaned of sensor error.

5 Model sets

The goal of this study was to evaluate the ability of PointNet to recognise primitive shapes in point clouds generated from scans of real objects, after being trained on sample point clouds of geometric shapes. All the point clouds were normalised before being fed into the PointNet: the normalisation procedure shrank or enlarged the shape without deformation to fit it into a box of side 1. The procedure described in Sect. 4.1 was run to extract numerical features from the scenes for the SNNs.

At present, there is no specific model set in the literature for benchmarking the accuracy of primitive shape classifiers on real-life scenes. For this study, a popular benchmark of 3D models of real-life objects was used: the YCB model set [12]. This model set was originally created and mainly used for robotic manipulation, instead of classification purposes. A subset of twenty-eight models from the YCB set was retained, containing objects of three basic primitive shapes: boxes, cylinders, and spheres. This subset was used to generate a test set of examples that will be henceforth called the *YCB-28* model set. In all the experiments, the *YCB-28* model set was employed to validate the learning accuracy of the trained neural networks. The methodology followed to create the *YCB-28* model set is detailed in Sect. 5.1. To train the classifiers, two artificial model sets were used. They are presented in Sects. 5.2 and 5.3. The normalisation procedure for point clouds before being fed into PointNet is described in Sect. 5.4

Table 1 IDs and names of the selected twenty-eight objects from the YCB set

Box	Cylinder	Sphere
003-cracker box	001-chips can	012-strawberry
004-sugar box	002-master chef can	014-lemon
008-pudding box	005-tomato soup can	015-peach
009-gelatin box	007-tuna fish can	017-orange
010-potted meat can	019-pitcher base	018-plum
026-sponge	025-mug	054-softball
036-wood block	040-large marker	055-baseball
061-foam brick	065-a-cups	056-tennis ball
077-rubiks cube		057-racquetball
		058-golf ball
		063-a-marble

5.1 The Yale-CMU-Berkeley (YCB) object and model set

The Yale-CMU-Berkeley object and model set was created by Calli et al. [12] for research in robotic manipulation. Calli et al. [12] used two series of depth cameras (BigBIRD Object Scanning Rig and Google Scanners) to capture point clouds from several real-life objects from multiple angles of views. The point clouds captured from each object were merged and de-noised to create mesh models, using the truncated signed distance function method [43] and Poisson reconstruction [44]. Only one mesh model was created for each object [12]. The mesh models were used for the experiments presented in this paper.

Differently from large classification sets like ModelNet40 [15], which contains 12,311 items from 40 different categories, the YCB set contains point clouds and mesh models from only 77 daily-life objects. These objects were

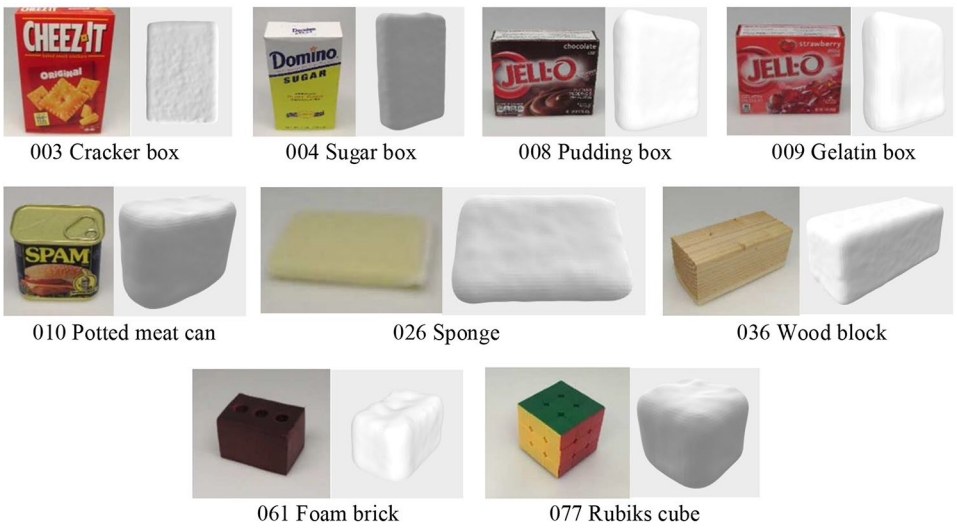

Fig. 2 The nine selected box-like objects (images and meshes) from YCB set

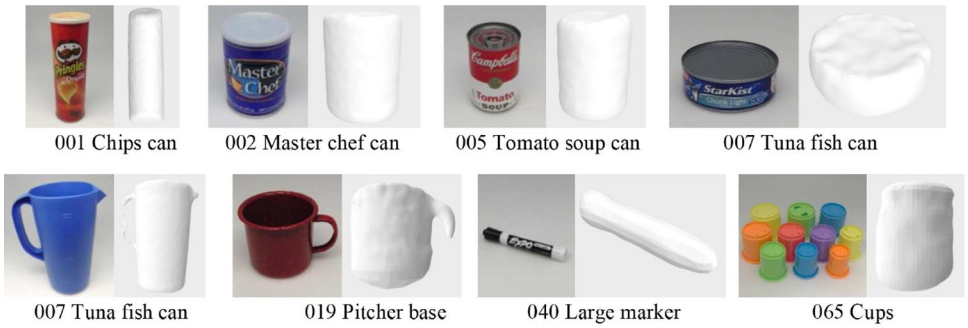

Fig. 3 The eight selected cylinder-like objects (images and meshes) from YCB set

broadly classified by Calli et al. [12] in 5 main categories: food items, kitchen items, tool items, shape items, and task items. In this study, the objects were grouped by their shape, and samples of appearance reasonably close to boxes, cylinders, and spheres were picked. The twenty-eight selected samples included nine models of box-shaped objects, eight models of cylinder-shaped objects, and eleven models of sphere-shaped objects.

The names and IDs (progressive identification number in the YCB set) of the twenty-eight selected models are reported in Table 1, and their pictures and mesh models are shown in Figs. 2, 3, and 4. Despite the de-noising and reconstruction, the mesh models still contain a certain level of sensor error, which is visible as irregularities such as bumps and hollows in the figures.

The following three-step procedure was used to generate the *YCB-28* model set out of the twenty-eight selected object scans. The first step was to randomly sample with uniform probability *1,000,000* points from the mesh model of each object. This initial large point set was called the *point pool*. The second step was to create *20* point clouds by randomly sampling *1000* out of the *1,000,000* points from the *point pool*. Each of the *20* point clouds created from one object model contained a different sample of points. Given that the sampling rate was (*1/1000*), it is reasonable to think that any two of the *20* point clouds had very little sampled points in common. Finally, in the third and last step, each point cloud was centred on the origin and the shape randomly rotated (roll-pitch-yaw rotation). In detail, the *YCB-28* model set contained the following point clouds:

- *Box:* 9 *objects* × 20 *pointclouds* = 180 *pointclouds*
- *Cylinder:* 8 *objects* × 20 *pointclouds* = 160 *pointclouds*
- *Sphere:* 11 *objects* × 20 *pointclouds* = 220 *pointclouds*
- *Total:* 560 *pointclouds*

In summary, the *YCB-28* model set contains 560 point clouds sampled from twenty-eight mesh models generated from real scenes, and was created for final performance testing.

5.2 The artificial primitive shapes (APS) sets

The APS set was used to train the neural network classifiers. This model set was originally created by Baronti et al. [45] for research on primitive shape fitting. The shape generation software can be downloaded from Baronti's GitHub repository[1].

The APS set contains point cloud models of the following three geometric shapes: *box*, *cylinder*, and *sphere*. Baronti et al. [45] created 591 different artificial shapes by changing the height (H), width (W), breadth (B), and diameter (D) of the three geometric shapes. The APS set was created from a full-factorial combination of the parameters defining each shape. The height, width, and breadth of the shapes were incremented from 1 to 10 in steps of 1 units. The diameter of the base of the cylinders was incremented from 0.5 to 5 in steps of 0.25 units. The diameter of the spheres was incremented from 1 to 10 in steps of 0.05 units. In summary:

- *Box*: 220 *point clouds with* $H,\ W,\ B \in \{1, 2, ..., 10\}$ $(H \geq W \geq B)$
- *Cylinder*: 190 *point clouds with* $D \in \{0.5, 0.75, ..., 5\}$ *and* $H \in \{1, 2, ..., 10\}$
- *Sphere*: 181 *point clouds with* $D \in \{1, 1.05, 1.10, ..., 10\}$
- *Total:* 591 *point clouds*

Given that the point clouds represent artificial objects, and are normalised before being fed to the PointNet, the unit of measurement of the H, W, B, and D parameters was not specified. It should also be noted that information on the size is not relevant to the determination of the shape of an object. All the shapes were placed with their centres at the origin of a Cartesian coordinate frame, and randomly oriented. The point clouds of the APS model set represent perfect shapes, since they are not corrupted by any sensor error. They are also complete, as opposed to real-life scans, like those of the YCB-28 set, where at least the information of base is missing since not reachable by the scanners. This set of perfect shapes will be henceforth called *APS-clean*.

[1] https://github.com/lucabaronti/BA-Primitive_Fitting_Dataset

Fig. 4 The eleven selected sphere-like objects (images and meshes) from YCB set

Baronti's software allows also injecting error (local imprecision simulating sensor inaccuracy) into the point clouds, as shown in Fig. 5. For details of the procedure used to inject error in the scenes, the reader is referred to [45]. A new model set was created duplicating the elements of the *APS-clean* set, normalising them within a bounding box of side 1, and for each element perturbing the position of the points of an amount randomly drawn with uniform probability within the interval [−0.025, +0.025]. This new model set will be henceforth called *APS-error*

Finally, a validation set containing 200 point clouds for each primitive shape was created. These shapes had random dimensions (H, W, B, D) and contained no sensor error. Henceforth, this set will be called *APS-clean-val*.

In summary, three model sets were created for the experiments from the original *APS* set: *APS-clean* and *APS-error* for training the classifiers, and *APS-clean-val* for optimisation of the SNNs and DNNs.

5.3 YCB-similar artificial primitive shapes

For each primitive shape, the *APS* set contained a wide range of shape variations. To make the training of the classifiers more focused on the recognition of the shapes of the *YCB-28* objects, one more model set of artificial primitive shapes was created. This model set contained artificial primitive shapes of features (H, W, B, D) more similar to those of the *YCB-28* objects. This new set allowed simulating the case where some knowledge about the expected shape of the objects is available.

Specifically, the Open3D open-source library [46] was obtained to enquire the shape features from the mesh models of the twenty-eight objects selected from the YCB set. Each mesh model was firstly visualised using Open3D. The shape features (H, W, B, D) of the objects were measured from the coordinates of manually picked key points from the visualised model. Three examples of the manually measured shape features are shown in Table 2, and the three objects and their mesh models are shown in Fig. 6, whilst the complete list of all the measured shape features from the twenty-eight selected YCB mesh models is detailed in Appendix 1. It should be noted that all point clouds are normalised in size before being fed to the classifiers. Therefore, the important information in the figures in Appendix 1 is not in the size but the proportions of the features.

Afterwards, twenty point clouds were generated from each of the twenty-eight selected objects based on their measured shape features. To generate a new point cloud,

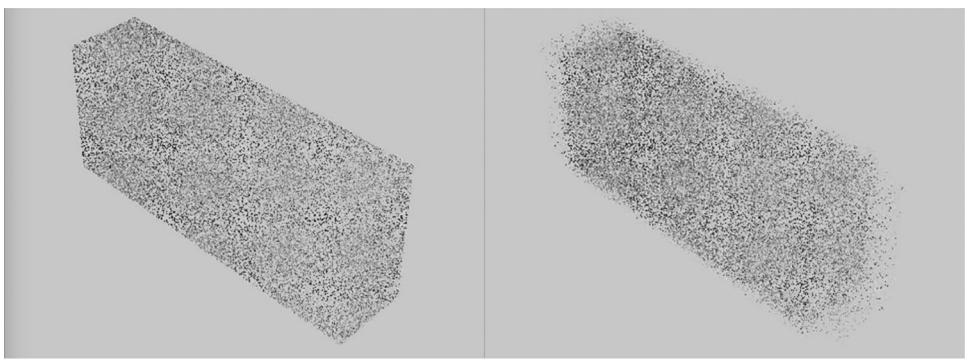

Fig. 5 Example of created box-shape point cloud, from left to right: point cloud without any error, and point cloud with error [45]

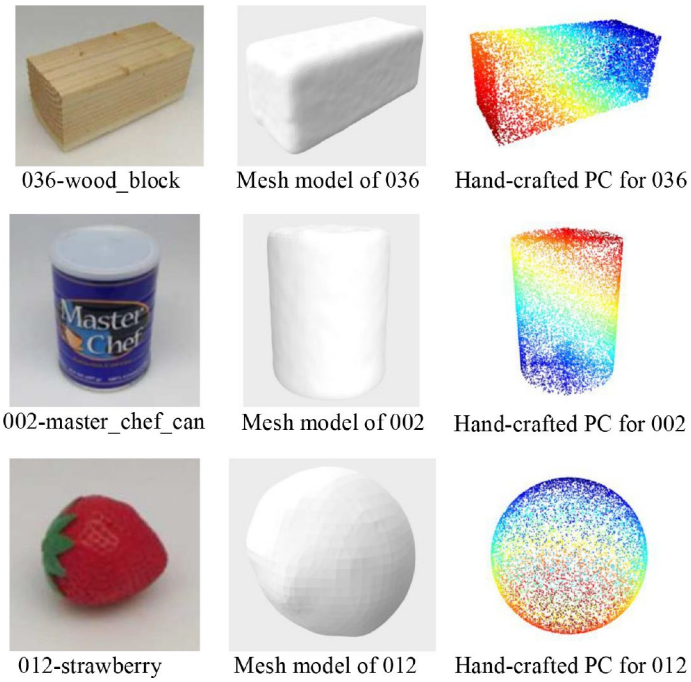

Fig. 6 Original YCB objects, their mesh models, and point cloud models of similar shape

each shape feature (H, W, B, D) was independently modified of a random amount. That is, each feature $K \in (H, W, B, D)$ of the primitive shape was randomly changed of an amount within the $[-5\%, +5\%]$ range of its size:

$$K' = [1 + x \sim U(-0.05, +0.05)] \times K \tag{1}$$

where $x \sim U(-0.05, +0.05)$ is a randomly sampled number from the uniform distribution $U(-0.05, +0.05)$. Consequently, each element of the new set of generated point clouds was similar in shape, although not the same, to the twenty-eight selected objects in *YCB-28*. Figure 6 shows examples of artificial point clouds near the real-life objects from which they were generated.

Henceforth, this new model set will be named *YCB-similar*. Its structure is as below:

- *Box*: 9 *objects* × 20 *pointclouds* = 180 *pointclouds*
- *Cylinder*: 8 *objects* × 20 *pointclouds* = 160 *pointclouds*
- *Sphere*: 11 *objects* × 20 *pointclouds* = 220 *pointclouds*
- *Total*: 560 *pointclouds*

Table 2 Manually measured shape features of three sample mesh models from *YCB-28* set

ID-name	Shape	H (mm)	W (mm)	B (mm)	D (mm)
036-wood block	Box	190.35	80.52	75.81	-
002-master chef can	Cylinder	129.79	-	-	96.09
012-strawberry	Sphere	-	-	-	48.52

In summary, the point cloud models of the *YCB-similar* set simulate the shape of objects in the *YCB-28* set. They contain no sensor error, and will be used for training the classifiers.

5.4 Point cloud normalisation

The point clouds were rescaled into a size 1 bounding box before they were fed to the PointNet. The procedure consists of the following steps. Given a point cloud $\mathcal{PC}$ containing N points, each point $\mathbf{p_i}$ is represented by coordinates (x_i, y_i, z_i) in a Cartesian 3D frame F:

$$\mathcal{PC} = \{\mathbf{p_1}, \mathbf{p_2}, \dots \mathbf{p_i}, \dots \mathbf{p_N}\} \tag{2}$$

$$\mathbf{p_i} = (x_i, y_i, z_i),\ i \in [1, N] \tag{3}$$

An initial rigid transformation is made so as all points p_i are described by positive (x_i, y_i, z_i) coordinate values:

$$\mathcal{PC}' = \{\mathbf{p_i'}\} = \{\mathbf{p_i} - \mathbf{p_{min}}\},\ i \in [1, N] \tag{4}$$

where $\mathbf{p_{min}} = (x_{min}, y_{min}, z_{min})$ and

$$x_{min} = \min_{\forall i \in [1,N]} x_i,\ y_{min} = \min_{\forall i \in [1,N]} y_i,\ z_{min} = \min_{\forall i \in [1,N]} z_i \tag{5}$$

The point cloud is then scaled based on the diagonal of its bounding box, and limited within the interval $x \in [0, 1],\ y \in [0, 1],\ z \in [0, 1]$:

$$\mathcal{PC}'' = \{\mathbf{p_i''}\} = \left\{\frac{\mathbf{p_i'}}{s}\right\},\ i \in [1, N] \tag{6}$$

where

$$s = \sqrt{(x'_{max})^2 + (y'_{max})^2 + (z'_{max})^2} \tag{7}$$

and

$$x'_{max} = \max_{\forall i \in N}(x'_i),\ y'_{max} = \max_{\forall i \in N}(y'_i),\ z'_{max} = \max_{\forall i \in N}(z'_i), \tag{8}$$

Finally, the point cloud is again translated so as its centroid (C) coincides with the origin of the Cartesian frame F:

$$\mathbf{C} = (x_c, y_c, z_c) = \left(\frac{1}{N}\sum_{i=0}^{N}(x''_i), \frac{1}{N}\sum_{i=0}^{N}(y''_i), \frac{1}{N}\sum_{i=0}^{N}(z''_i)\right) \tag{9}$$

$$\tilde{\mathcal{PC}} = \{\mathbf{p''_i} - \mathbf{COG}\},\ i \in [1, N] \tag{10}$$

This procedure was introduced to resize the point clouds to a bounded space. It is important to notice that the normalisation does not rescale the objects to a standard size, since the size after normalisation depends not only on the view but also the orientation of the objects. For example, the size of a cuboid will be largest when all its sides are aligned to the coordinate axes, and smallest when one of its diagonals is aligned to the coordinate axes.

6 Experiments and results

This section describes the experimental setup and the results obtained by PointNet and the two SNN classifiers. Three artificial model sets, *APS-clean* model set, *APS-error* model set, and *YCB-similar* model set, were used for training the neural networks. The final performance of trained classifiers was evaluated on the *YCB-28* model set. In all experiments, 10 independent learning trials were performed, and the results were statistically analysed.

The PointNet architecture used in the experiments was obtained from open-source code made available by Qi et al. [6] in their Github repository[2]. Following the methodology of Qi et al. [6], the PointNet was trained using the Adam optimiser [47]. Adam updates the network weights based on gradients calculated from randomly picked mini-batches of point clouds of predefined size. The batch size is an important hyper-parameter of the algorithm.

Based on preliminary tests, the structure and most of the hyper-parameters of PointNet were kept as originally designed by Qi et al. [6], and the remaining hyper-parameters were manually optimised. They are shown in Table 3. The MLP was trained using the standard BP algorithm with momentum term [39], whilst the RBFN was trained using first a KNN-based algorithm for a broad brush optimisation of the radial basis function parameters, and then the BP algorithm to fine tune the whole network parameters. The hyper-parameters of the SNNs and those of their learning procedures were optimised by trial and error. They are detailed in Table 4.

Two hyper-parameters, the batch size used by Adam training procedure and the number of training epochs, have the largest effect on the learning accuracy of PointNet. Section 6.1 describes the procedure followed for their experimental optimisation. This procedure was carried out using the artificial *APS-clean* model set. After optimisation, since the PointNet was always tested on the same benchmark problem (*YCB-28*), the hyper-parameters were fixed for all the experiments. The results of the experiments are reported in Sect. 6.2.

Table 3 Hyper-parameters of PointNet used in the experiments

Parameter	Value
Training algorithm	Adam
Decay rate β_1 in Adam	0.9
Decay rate β_2 in Adam	0.999
Numerical constant $\hat{\epsilon}$ in Adam	1×10^{-7}
Initial learning rate	0.0001
Minimum learning rate	0.00001
Learning rate decay rate	0.7
Learning rate decay step	200,000
Initial batch normalisation momentum	0.5
Maximum batch normalisation momentum	0.99
Batch normalisation momentum decay rate	0.5
Batch normalisation momentum decay step	200,000

6.1 Optimisation of the PointNet training procedure

PointNet was trained using the *APS-clean* model set, and the results validated using *APS-clean-val* set. That is, the DNN was trained and optimised using only knowledge from perfect artificial shapes.

Table 4 Hyper-parameters of MLP and RBFN used in the experiments

Parameter	MLP	RBFN
Training algorithm	Back propagation	Back propagation
Number of inputs	6	6
Number of hidden layers	1	1
Number of hidden neurons	15	20
Number of outputs	3	3
Learning rate	0.01	0.01
Training epoch	2000	5000

[2] https://github.com/charlesq34/pointnet

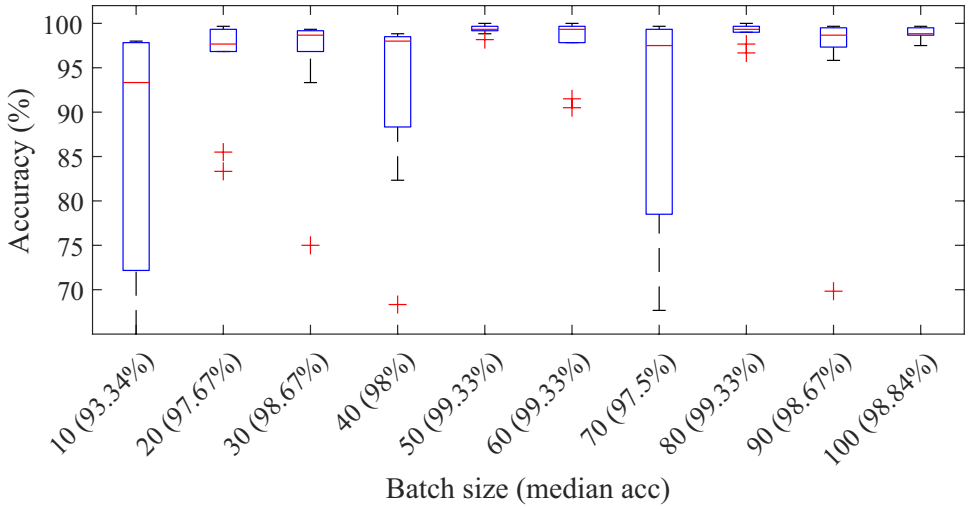

Fig. 7 Performance of PointNet as the batch size setting is varied. The DNN was trained using the *APS-clean* and the learning accuracy tested on the *APS-val* model set

6.1.1 Batch size optimisation

To optimise the batch size used by the Adam optimiser, tests were performed varying the hyper-parameter from 10 to 100 in steps of 10, fixing the number of training epochs to 200. Ten learning trials were performed for each batch size setting.

The experimental results are shown using box plots in Fig. 7. The red line within the box indicates the median result of the 10 independent learning trials. In terms of median accuracy, the performance improves noticeably when the batch size is increased from 10 (93.34% median accuracy) to 20 (97.67%), and from 40 (98.0%) to 50 (99.33%). Both improvements are statistically significant at an $\alpha = 0.01$ confidence level: the p-value is 0.0343 for the difference between the results obtained using batch sizes of 10 and 20, and $p = 0.0006$ for the difference between the results obtained using batch sizes of 40 and 50.

Beyond a size of 50, the statistical analysis suggests there are no benefits in any further increase of the batch size. However, as the batch size increases, the training procedure appears to become more consistent (smaller width of the box plots), and for this reason a batch size 100 was chosen. Although this choice is the most computationally intensive, the training process can be sped up using GPU acceleration.

Since the number of PCs constituting the model sets used in the experiments is not a multiple of 100, the number of training examples in the last batch fed to the *Adam* optimiser had to be brought to 100. This was achieved by duplicating randomly picked PCs from the whole model set. For example, the *APS-error* set contained 591 PCs, which were fed in 6 batches of 100 PCs each. The last batch was formed by the remaining unused 91 examples, and 9 randomly picked duplicates.

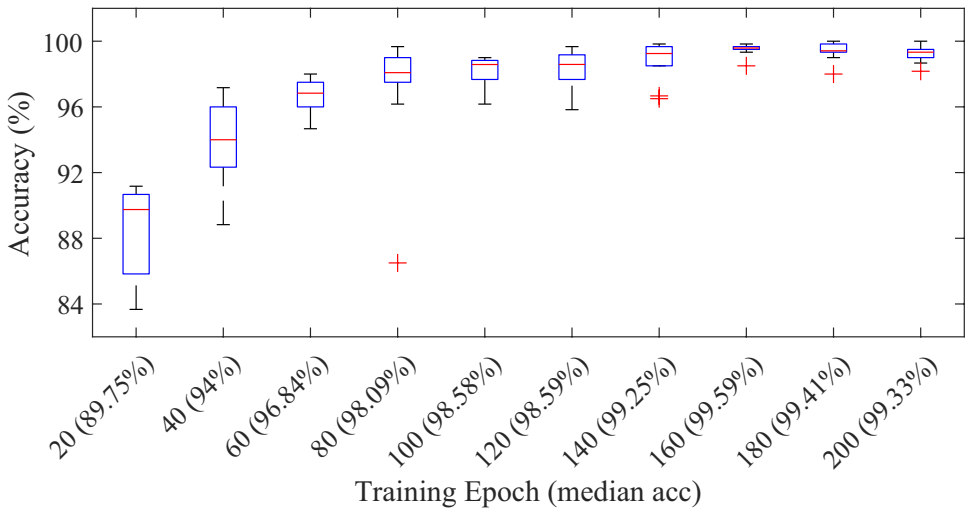

Fig. 8 Performance of PointNet as the number of training epochs is varied. The DNN was trained using the *APS-clean* and the learning accuracy tested on the *APS-val* model set

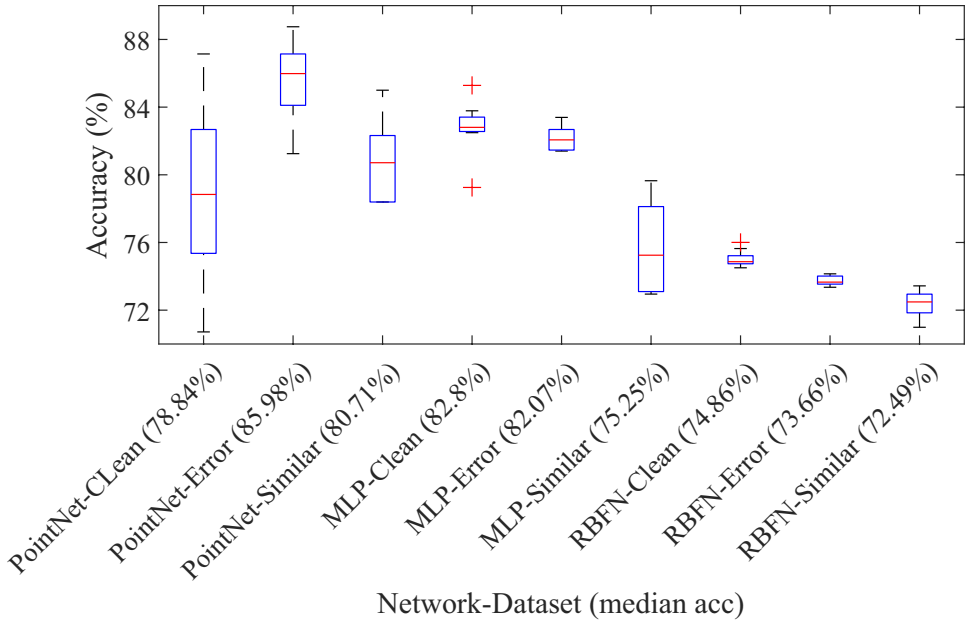

Fig. 9 Accuracy results obtained on the *YCB-28* model set by the three classifiers when trained using different sets: *APS-clean* (Clean), *APS-error* (Error) and *YCB-similar* (Similar)

6.1.2 Training epoch optimisation

After the batch size had been fixed to 100, the number of training epochs was optimised by trial and error. Experiments were performed increasing the number of epochs from 20 to 200 in steps of 20. The results are shown in Fig. 8.

Figure 8 shows a progressive improvement in the performance of PointNet as the number of training epochs is increased until 160. Pairwise Mann-Whitney statistical tests indicated that the performance of PointNet trained using 160 epochs is significantly superior to the performance obtained using any smaller number of training epochs (from 20 to 140 epochs). Further increases of the number of training epochs beyond the 160 did not yield any significant improvement in performance. Consequently, this hyper-parameter was fixed to 160.

6.2 Experimental results — artificial model sets

As mentioned in Sect. 6, three instances of PointNet were trained using three artificial model sets: *APS-clean*, *APS-error*, and *YCB-similar*, using the hyper-parameters shown in Table 3 and discussed in Sect. 6.1. The performance of the trained PointNets was evaluated on their accuracy on the *YCB-28* model set, and compared to that of two SNNs: an MLP and an RBFN. The results of the experiments are visualised using box plots in Fig. 9, and fully detailed in Table 5. Each box in the figure visualises the five-number summary of the accuracy results attained in the 10 independent learning trials. The significance of the differences in the results obtained in the various sets of learning trials was evaluated via pairwise Mann-Whitney tests. Table 6 fully details the results of the significance tests.

Table 5 Shape identification accuracies obtained on the *YCB-28* model set by the three neural network architectures. Each column reports the results of 10 independent learning trials performed using a different training set: *APS-clean* (Clean), *APS-error* (Error), and *YCB-similar* (Similar)

Index	PointNet			MLP			RBF		
	Clean	Error	Similar	Clean	Error	Similar	Clean	Error	Similar
1	84.64%	85.54%	82.32%	83.78%	82.28%	74.42%	74.93%	73.35%	72.70%
2	75.00%	84.11%	64.46%	82.56%	82.32%	76.08%	74.50%	73.54%	71.48%
3	82.68%	81.25%	81.25%	82.69%	81.40%	79.65%	74.51%	73.35%	72.95%
4	79.11%	87.32%	80.18%	79.25%	81.40%	73.10%	75.17%	73.84%	70.99%
5	81.79%	81.79%	85.00%	82.86%	81.65%	78.12%	74.80%	73.72%	73.19%
6	75.36%	85.54%	82.14%	82.74%	81.85%	72.95%	75.22%	74.01%	72.95%
7	70.71%	88.75%	45.00%	83.41%	83.39%	76.72%	76.01%	73.60%	72.03%
8	87.14%	86.43%	78.39%	85.28%	82.77%	78.41%	74.74%	73.60%	73.44%
9	78.57%	87.14%	78.75%	83.29%	81.46%	73.26%	74.74%	74.03%	72.27%
10	76.96%	86.79%	82.68%	82.49%	82.68%	72.95%	75.64%	74.15%	71.84%
Median	78.84%	***85.98%***	80.71%	82.80%	82.07%	75.25%	74.86%	73.66%	72.49%

Table 6 Mann-Whitney test results for each pair of experiments. Results equal to or below the $\alpha = 0.01$ confidence level are reported in bold

		PointNet			MLP			RBF		
		Clean	**Error**	**Similar**	**Clean**	**Error**	**Similar**	**Clean**	**Error**	**Similar**
PointNet	**Clean**	-	**0.007**	0.910	0.041	0.162	0.059	**0.010**	**0.002**	**0.002**
	Error	-	-	**0.002**	0.019	**0.010**	**0.000**	**0.000**	**0.000**	**0.000**
	Similar	-	-	-	**0.010**	0.112	0.041	0.023	0.023	0.023
MLP	**Clean**	-	-	-	-	0.034	**0.000**	**0.000**	**0.000**	**0.000**
	Error	-	-	-	-	-	**0.000**	**0.000**	**0.000**	**0.000**
	Similar	-	-	-	-	-	-	1.000	0.450	**0.001**
RBF	**Clean**	-	-	-	-	-	-	-	**0.000**	**0.000**
	Error	-	-	-	-	-	-	-	-	**0.000**
	Similar	-	-	-	-	-	-	-	-	-

Figure 9 shows that PointNet achieved an average (median) accuracy of 86% circa when trained on the *APS-error* model set. If a stringent $\alpha = 0.01$ confidence level is sought, Table 6 indicates that when trained on the *APS-error* model set, PointNet outperformed any other combination of classifier and training set, except for the MLP trained on the clean set. If the confidence level is relaxed to the often used $\alpha = 0.05$, it can be said that PointNet trained on the *APS-error* set was the clear winner of the comparison.

PointNet did not perform equally well when trained using the *APS-clean* and *APS-similar* model sets, although the performance on the latter (81% circa average accuracy) was still adequate.

Despite the unsophisticated feature extraction method used, the MLP performed remarkably well. Trained using the *APS-clean* model set, it achieved nearly 83% training accuracy, and slightly less (82%) when trained using the *APS-error* set. Compared to the PointNet, the MLP obtained more consistent learning results, as shown by the width of the box plots. The RBFN was the clear underperformer of the three tested classifiers. Finally, training the classifiers on the *YCB-Similar* set did not provide any visible benefit, particularly for the two SNNs.

7 Discussion

In this study, the hyper-parameters of PointNet were tuned using artificial models. The exercise can be seen as an attempt to evaluate whether knowledge gained on artificial models could be transferred to real-life scenes. The study aimed also at testing whether simple SNNs were able to obtain results comparable to those obtained by the much more complex PointNet.

The experimental tests showed that, in terms of accuracy, PointNet had indeed an edge, albeit small, on a standard shallow MLP classifier. However, the MLP showed more consistent training results. The tests also indicated that PointNet performs best (85.98%) when trained on scenes that were perturbed with some level of random error. Training PointNet using shapes of features similar to those to the real images, instead of training it with more general artificial shapes, did not significantly improve the accuracy of the classifier.

The above results suggest that the generalisation accuracy of PointNet is likely to be sensitive to sensor error. As the mesh models in Figs. 2, 3, and 4 show, this error had not been completely removed by the pre-processing procedures. Trained for 160 epochs on perfect artificial shapes, PointNet was able to obtain nearly perfect recognition accuracy (99.59%) on previously unseen artificial shapes (Fig. 8). However, the average accuracy of PointNet dropped to 78.84% when the trained network was tasked with recognising the shape of real-life objects. The learning results were also not consistent, and widely varied in quality between the ten independent learning trials performed (Table 5).

Training PointNet on models of features that are closer to those of the real-life objects (*YCB-Similar*) produced only a marginal improvement in accuracy and consistency, because the *YCB-Similar* models still consist of clean geometric shapes. However, training PointNet using noisy scenes (*APS-error*) markedly improved its learning accuracy and consistency. That is, training the DNN to recognise 'imperfect' shapes increased its ability to correctly classify shapes from imperfect real-life scans.

The difficulties encountered by PointNet to generalise the knowledge learned from artificial models to models of real-life objects have been already reported by other authors [14]. These difficulties are common to the general DNN field, where large over-parameterised structures are often able to perfectly fit the training data, and have issues of poor generalisation or overfitting [48, 49].

A contribution from this study is the confirmation of the validity of the idea of injecting local imprecision in the training shapes, so as to 'blur' their boundaries and prevent PointNet from learning the perfect examples. This approach bears similarities with data augmentation techniques where slightly modified copies of already existing data are added to the training set, in order to regularise the

neural network models [50]. Other similar regularisation procedures contaminate the input patterns with randomly re-sampled noise at each iteration of the learning procedure [51, 52]. The common approach of all these procedures is to oversample the training set to smoothen the mapping of the neural network model. Their common goal is to optimise the bias-variance trade-off of the learned model, and promote generalisation [51]. Rather than smoothening PointNet mapping, the approach used in this study aims to promote a 'tolerance' to local imprecision, similar to the approximation threshold used in the RANSAC algorithm [53]. The proposed approach also does not augment the training data or re-sample the noise at every iteration, promoting thus the efficiency of the learning procedure.

It should be noted that both the SNNs obtained their best learning accuracies when trained on the *APS-clean* set. This result might be due to the particular feature extraction method used (Sect. 4.1), where the simulated sensor error might have excessively blurred the shape of the projections of some objects. It may also indicate that the SNNs did not overfit the training data. In general, trained on the *APS-clean* set, the MLP obtained higher classification accuracies and more consistent results than PointNet.

The main advantage of PointNet is that the data requires only minimal pre-processing (normalisation and down sampling), beyond the standard cleaning of the raw point cloud scenes. In particular, PointNet does not need the feature extraction process required by the MLP. The feature extraction process is embedded in the first block of layers of the PointNet, and optimised simultaneously to the classifier by the learning algorithm. In this study, the feature extraction process was carried out prior to the MLP training procedure, and it is possible that the criterion of the former did not perfectly match the inductive and representational biases of the latter. The concurrency of the feature extraction and classification procedures might have given and edge to the PointNet respect to the MLP.

7.1 Indications for future work

This study proved that PointNet can be trained using artificial data to recognise with good accuracy shapes from real-life scans of objects. This approach makes it easier for designers to build the usually large data set needed to train the classifier.

The experimental work was based on the recognition of three primitive shapes: box, cylinder, and sphere. Further work should validate the proposed method on a more varied and complex set of objects. In particular, given the context of robotic disassembly, the proposed approach should be validated on models of real mechanical parts. Although preliminary tests suggested the applicability of the technique to complex automotive components [7], the simulations did not take into account real-world occurrences such as reflective surfaces.

The main hurdle to an extensive testing of the proposed approach has been so far the lack of a database of real-life mechanical object models. The assembly of such set has been hampered by the restrictions due to the recent pandemic. The collection and scanning of object samples is now a priority.

PointNet has shown a tendency to overfit the training data, showing poor generalisation capability on noisy data. Addition of noise to the training samples boosted the performance of PointNet. Further work should be done to test other regularisation techniques such as dropout [54] and weight decay [55].

The generalisation ability of PointNet could also be improved by decreasing the complexity of its architecture. Based on preliminary tests, this architecture has been kept so far similar to the one originally designed by Qi et al. [6] (see Sect. 6). A more thorough analysis might reveal the advantage of more economical structures.

Finally, the segmentation ability of PointNet should also be explored. Scene segmentation will be very useful in disassembly scenarios to identify end-of-life products and their sub-assemblies in scanned scenes.

8 Conclusions

This study investigated the possibility of training the PointNet DNN on point cloud models of perfect geometric primitive shapes, and use it to recognise primitive shapes in models of daily-life objects. The ultimate objective of the study is to use PointNet to generate shape information for robotic manipulation and disassembly of end-of-life products.

Experimental tests showed that, trained on perfect geometric shapes, PointNet was able to recognise with nearly 80% average accuracy primitive shapes in real-life objects. The tests also showed some inconsistency in the performance of the DNN. Trained on perfect geometric shapes using a simple feature extraction method, a simple shallow MLP architecture obtained better results than the PointNet in terms of average accuracy and consistency of the learning results. The main difficulty found by PointNet seemed to consist of generalising the knowledge gained on perfect artificial shapes to real-life cases. This finding confirmed the results of a handful of similar experiments in the literature, and is the first contribution of this study.

The accuracy of PointNet could be raised to nearly 86% by locally perturbing the position of the elements of the training point clouds. This operation corresponded to blurring the representation of the shapes, in order to train the DNN on imprecise models that are more similar to real-life representations. In this study, this new training method has been verified on the recognition of shapes from real-life objects. In addition to improving the recognition accuracy, it also greatly improved the consistency of PointNet learning results. This result constitutes the second contribution of this study.

Indications for further work were given in Sect. 7.

Appendix

Features of *YCB-28* objects

The main features of the YCB-28 objects are described in Table 7.

Table 7 Manually measured shape features of the twenty-eight selected objects from YCB set

ID-name	Shape-type	H (mm)	W (mm)	B (mm)	D (mm)
003-cracker box	Box	216.23	149.82	57.04	-
004-sugar box	Box	162.50	82.58	32.32	-
008-pudding box	Box	97.80	79.34	29.38	-
009-gelatin box	Box	73.70	65.62	23.73	-
010-potted meat can	Box	80.16	76.26	39.73	-
026-sponge	Box	95.09	60.03	14.20	-
036-wood block	Box	190.35	80.52	75.81	-
061-foam brick	Box	63.68	54.29	43.36	-
077-rubiks cube	Box	57.35	48.56	46.12	-
001-chips can	Cylinder	237.27	-	-	69.33
002-master chef can	Cylinder	129.79	-	-	96.09
005-tomato soup can	Cylinder	92.34	-	-	64.63
007-tuna fish can	Cylinder	38.44	-	-	80.25
019-pitcher base	Cylinder	226.75	-	-	128.13
025-mug	Cylinder	101.24	-	-	84.78
040-large marker	Cylinder	103.09	-	-	14.81
065-a-cups	Cylinder	57.36	-	-	43.53
012-strawberry	Sphere	-	-	-	48.52
014-lemon	Sphere	-	-	-	63.59
015-peach	Sphere	-	-	-	73.05
017-orange	Sphere	-	-	-	74.81
018-plum	Sphere	-	-	-	52.29
054-softball	Sphere	-	-	-	97.03
055-baseball	Sphere	-	-	-	75.46
056-tennis ball	Sphere	-	-	-	69.18
057-racquetball	Sphere	-	-	-	60.41
058-golf ball	Sphere	-	-	-	47.36
063-a-marble	Sphere	-	-	-	39.92

Author contribution All authors contributed to the study conception and design. Data collection and preparation and the experimental tests were performed by Senjing Zheng, data analysis by both authors. The first draft of the manuscript was written by Senjing Zheng and Marco Castellani edited it and expanded it. All authors read and approved the final manuscript.

Funding This work was supported by the UK Engineering and Physical Sciences Research Council (EPSRC) granted Autonomous Remanufacturing (AutoReman) project (Grant No. EP/N018524/1).

Declarations

Competing interests The authors declare no competing interests.

References

1. Kopicki M, Detry R, Adjigble M, Stolkin R, Leonardis A, Wyatt JL (2016) One-shot learning and generation of dexterous grasps for novel objects. Int J Robot Res 35(8):959–976
2. Saxena A, Driemeyer J, Ng AY (2008) Robotic grasping of novel objects using vision. Int J Robot Res 27(2):157–173
3. Mavrakis N, Stolkin R, Baronti L, Kopicki M, Castellani M et al (2016) Analysis of the inertia and dynamics of grasped objects, for choosing optimal grasps to enable torque-efficient post-grasp manipulations. In: 2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids), IEEE, pp 171–178
4. Vongbunyong S, Kara S, Pagnucco M (2013) Application of cognitive robotics in disassembly of products. CIRP Ann 62(1):31–34
5. Wegener K, Chen WH, Dietrich F, Dröder K, Kara S (2015) Robot assisted disassembly for the recycling of electric vehicle batteries. Procedia Cirp 29:716–721
6. Qi CR, Su H, Mo K, Guibas LJ (2017) PointNet: deep learning on point sets for 3D classification and segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp 652–660
7. Zheng S, Lan F, Baronti L, Pham DT, Castellani M (2022) Automatic identification of mechanical parts for robotic disassembly using the PointNet deep neural network. Int J Manuf Res 17(1):1–21
8. Rabbani T, Van Den Heuvel F (2005) Efficient Hough transform for automatic detection of cylinders in point clouds. Isprs Wg Iii/3, Iii/4 3:60–65
9. Zheng Y, Liu J, Liu Z, Wang T, Ahmad R (2019) A primitive-based 3D reconstruction method for remanufacturing. Int J Adv Manuf Technol 103(9):3667–3681
10. Pham D, Liu X (1995) Neural Networks for Identification. Springer, Prediction and Control

11. Broomhead DS, Lowe D (1988) Radial basis functions, multivariable functional interpolation and adaptive networks. Tech. rep, Royal Signals and Radar Establishment Malvern (United Kingdom)
12. Calli B, Singh A, Walsman A, Srinivasa S, Abbeel P, Dollar AM (2015) The YCB object and model set: towards common benchmarks for manipulation research. In: 2015 International Conference on Advanced Robotics (ICAR), IEEE, pp 510–517
13. Garcia-Garcia A, Orts-Escolano S, Oprea S, Villena-Martinez V, Garcia-Rodriguez J (2017) A review on deep learning techniques applied to semantic segmentation. arXiv preprint arXiv:1704.06857
14. Uy MA, Pham QH, Hua BS, Nguyen T, Yeung SK (2019) Revisiting point cloud classification: a new benchmark dataset and classification model on real-world data. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp 1588–1597
15. Wu Z, Song S, Khosla A, Yu F, Zhang L, Tang X, Xiao J (2015) 3D shapenets: a deep representation for volumetric shapes. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp 1912–1920
16. LeCun Y, Bengio Y, Hinton G, et al. (2015) Deep learning. nature, 521 (7553), 436-444. Google Scholar Google Scholar Cross Ref Cross Ref
17. Börold A, Teucke M, Rust J, Freitag M (2020) Recognition of car parts in automotive supply chains by combining synthetically generated training data with classical and deep learning based image processing. Procedia CIRP 93:377–382
18. Krueger J, Lehr J, Schlueter M, Bischoff N (2019) Deep learning for part identification based on inherent features. CIRP Ann 68(1):9–12
19. Weimer D, Scholz-Reiter B, Shpitalni M (2016) Design of deep convolutional neural network architectures for automated feature extraction in industrial inspection. CIRP Ann 65(1):417–420
20. Brogan DP, DiFilippo NM, Jouaneh MK (2021) Deep learning computer vision for robotic disassembly and servicing applications. Array 12:100094
21. Foo G, Kara S, Pagnucco M (2021) Screw detection for disassembly of electronic waste using reasoning and re-training of a deep learning model. Procedia CIRP 98:666–671
22. Li X, Li M, Wu Y, Zhou D, Liu T, Hao F, Yue J, Ma Q (2021) Accurate screw detection method based on faster R-CNN and rotation edge similarity for automatic screw disassembly. Int J Comput Integr Manuf 34(11):1177–1195
23. Rehnholm J (2021) Battery pack part detection and disassembly verification using computer vision
24. Yildiz E, Wörgötter F (2019) DCNN-based screw detection for automated disassembly processes. In: 2019 15th International Conference on Signal-Image Technology & Internet-Based Systems (SITIS), IEEE, pp 187–192
25. Yildiz E, Wörgötter F (2020) DCNN-based screw classification in automated disassembly processes. In: ROBOVIS, pp 61–68
26. Ren S, He K, Girshick R, Sun J (2015) Faster R-CNN: towards real-time object detection with region proposal networks. Adv Neural Inf Process Syst 28:91–99
27. Maturana D, Scherer S (2015) VoxNet: a 3D convolutional neural network for real-time object recognition. In: 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, pp 922–928
28. Su H, Maji S, Kalogerakis E, Learned-Miller E (2015) Multi-view convolutional neural networks for 3D shape recognition. In: Proceedings of the IEEE International Conference on Computer Vision, pp 945–953
29. Klokov R, Lempitsky V (2017) Escape from cells: deep Kd-networks for the recognition of 3D point cloud models. In: Proceedings of the IEEE International Conference on Computer Vision, pp 863–872
30. Riegler G, Osman Ulusoy A, Geiger A (2017) OctNet: learning deep 3D representations at high resolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp 3577–3586
31. Wang PS, Liu Y, Guo YX, Sun CY, Tong X (2017) O-CNN: octree-based convolutional neural networks for 3D shape analysis. ACM Transactions On Graphics (TOG) 36(4):1–11
32. Guo Y, Wang H, Hu Q, Liu H, Liu L, Bennamoun M (2020) Deep learning for 3D point clouds: a survey. IEEE Trans Pattern Anal Mach Intell
33. Joseph-Rivlin M, Zvirin A, Kimmel R (2019) Momen (e) t: flavor the moments in learning to classify shapes. In: Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops, pp 0
34. Sun X, Lian Z, Xiao J (2019) SRINet: learning strictly rotation-invariant representations for point cloud classification and segmentation. In: Proceedings of the 27th ACM International Conference on Multimedia, pp 980–988
35. Yang J, Zhang Q, Ni B, Li L, Liu J, Zhou M, Tian Q (2019) Modeling point clouds with self-attention and Gumbel subset sampling. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp 3323–3332
36. Jaderberg M, Simonyan K, Zisserman A et al (2015) Spatial transformer networks. Adv Neural Inf Process Syst 28:2017–2025
37. Dominguez M, Dhamdhere R, Petkar A, Jain S, Sah S, Ptucha R (2018) General-purpose deep point cloud feature extractor. In: 2018 IEEE Winter Conference on Applications of Computer Vision (WACV), IEEE, pp 1972–1981
38. Andina D, Pham DT (2007) Computational intelligence: for engineering and manufacturing. Springer
39. Rumelhart DE, Hinton GE, Williams RJ (1985) Learning internal representations by error propagation. California Univ San Diego La Jolla Inst for Cognitive Science, Tech. rep
40. Hornik K, Stinchcombe M, White H (1989) Multilayer feedforward networks are universal approximators. Neural Netw 2(5):359–366
41. Hotelling H (1933) Analysis of a complex of statistical variables into principal components. J Educ Psychol 24(6):417
42. Pearson K (1901) LIII. On lines and planes of closest fit to systems of points in space. The London, Edinburgh, and Dublin philosophical Magazine and Journal of Science 2(11):559–572
43. Curless B, Levoy M (1996) A volumetric method for building complex models from range images. In: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, pp 303–312
44. Kazhdan M, Bolitho M, Hoppe H (2006) Poisson surface reconstruction. In: Proceedings of the Fourth Eurographics Symposium on Geometry Processing, vol 7
45. Baronti L, Alston M, Mavrakis N, Ghalamzan EAM, Castellani M et al (2019) Primitive shape fitting in point clouds using the bees algorithm. Appl Sci 9(23):5198
46. Zhou QY, Park J, Koltun V (2018) Open3D: a modern library for 3D data processing. arXiv preprint arXiv:1801.09847
47. Kingma DP, Ba J (2014) Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980
48. Bejani MM, Ghatee M (2021) A systematic review on overfitting control in shallow and deep neural networks. Artif Intell Rev pp 1–48
49. Salman S, Liu X (2019) Overfitting mechanism and avoidance in deep neural networks. arXiv preprint arXiv:1901.06566
50. Hernández-García A, König P (2018) Further advantages of data augmentation on convolutional neural networks. In: International Conference on Artificial Neural Networks, Springer, pp 95–103
51. Bishop CM (1995) Training with noise is equivalent to Tikhonov regularization. Neural Comput 7(1):108–116
52. Matsuoka K (1992) Noise injection into inputs in back-propagation learning. IEEE Trans Syst Man Cybern 22(3):436–440

53. Schnabel R, Wahl R, Klein R (2007) Efficient RANSAC for point-cloud shape detection. Computer graphics forum, Wiley Online Library 26:214–226
54. Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R (2014) Dropout: a simple way to prevent neural networks from overfitting. J Mach Learn Res 15(1):1929–1958
55. Krogh A, Hertz J (1991) A simple weight decay can improve generalization. Adv Neural Inf Proces Syst 4