Survey and Evaluation of Neural 3D Shape Classification Approaches

Martin Mirbauer, Miroslav Krabec, Jaroslav Křivánek, Elena Šikudová

Abstract—Classification of 3D objects – the selection of a category in which each object belongs – is of great interest in the field of machine learning. Numerous researchers use deep neural networks to address this problem, altering the network architecture and representation of the 3D shape used as an input. To investigate the effectiveness of their approaches, we conduct an extensive survey of existing methods and identify common ideas by which we categorize them into a taxonomy. Second, we evaluate 11 selected classification networks on two 3D object datasets, extending the evaluation to a larger dataset on which most of the selected approaches have not been tested yet. For this, we provide a framework for converting shapes from common 3D mesh formats into formats native to each network, and for training and evaluating different classification approaches on this data. Despite being partially unable to reach the accuracies reported in the original papers, we compare the relative performance of the approaches as well as their performance when changing datasets as the only variable to provide valuable insights into performance on different kinds of data. We make our code available to simplify running training experiments with multiple neural networks with different prerequisites.

Index Terms—3D shape analysis, classification algorithms, computer graphics, convolutional neural network, deep learning, image processing, machine learning, multi-layer neural network, neural networks, object recognition.

1 Introduction

Classification and generation of 3D shapes is one of the widely researched topics in the field of artificial intelligence. It is applied in a vast number of fields such as autonomous driving [1], analysis of medical data [2] as well as various fields of computer vision and graphics [3, 4]. Classification of objects in 2D images has been revolutionized by deep convolutional neural networks [5, 6] and has been shown to achieve super-human accuracy [7]. This is not yet the case for 3D shapes, perhaps because of the lack of a representation that is both expressive and easy to process by a neural network.

Numerous network architectures working with different 3D shape representations have been designed, and new ones are still being developed. However, their relative performance needs further evaluation and comparison.

As the number of published approaches increases, understanding existing approaches, finding the proper representation and approach for a given application, and following new ones becomes more difficult. Categorizing them into a taxonomy and comparing the methods which use different representations is essential to simplify orientation in the landscape of approaches.

In this work, we focus on supervised learning, specifically the classification task, which is closely related to global feature extraction – one of the tasks in the broader context of machine *understanding* of shapes and scenes.

We define the classification task as follows: we are given a set of training examples $\{(x_1, y_1), \ldots, (x_n, y_n)\}$, where x_i is a 3D shape representation and y_i is a numerical encoding of the corresponding label. Each shape belongs to exactly one class. A classification model is a parametric model $P(\theta) : X \to Y$, where X is a space of 3D shapes, Y is a space of labels, and θ are trainable parameters. With θ optimized to minimize a prediction error metric, $P(\theta)$ should predict the correct class label for each 3D shape from X.

The contributions of this work are:

First, we extensively survey deep learning-based 3D shape classification approaches published before October 2019 and categorize them based on common approach ideas, which provides researchers with an overview of approaches suitable for processing 3D shapes.

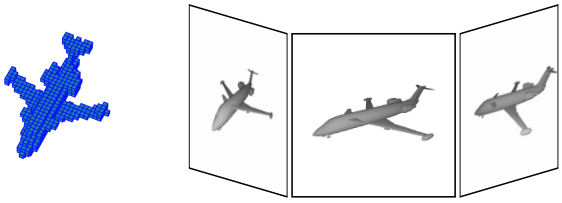
Second, we select several existing techniques of 3D shape classification to replicate their reported results, compare and evaluate them on publicly available CAD datasets. We provide a pipeline which simplifies evaluating quality of new classifiers and methods for converting between different shape representations. The code is available on the project's website[1].

1.1 Related Work

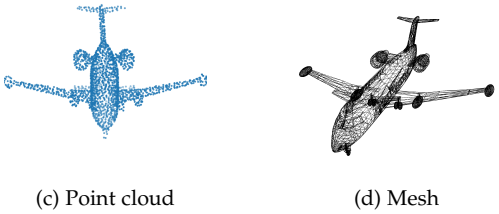
There are existing works surveying the machine learning methods which process 3D shapes, however Zelener [8], Ioannidou et al. [9] and Carvalho and von Wangenheim [10] survey publications before the year 2016, which we extend until the end of year 2020 thus including current state-of-the-art methods. Other works by Arnold et al. [11], Griffiths and Boehm [12] focus on processing scanned data (RGB-D

- *Manuscript received [TODO when] ... The work was supported by the Charles University Grant Agency project GAUK 966119. This work was supported by the Charles University grant SVV-260588. (Corresponding author: Martin Mirbauer)*
- *The authors are with Charles University, Faculty of Mathematics and Physics, Prague, Czech Republic.*
  E-mail: {martinm,sikudova}@cgg.mff.cuni.cz
  J. Křivánek, deceased, was also with Chaos Czech a.s., Prague, Czech Republic.

1. https://cgg.mff.cuni.cz/~martinm/papers/2021-survey-eval



(a) Volumetric grid (b) Multi-view



(c) Point cloud (d) Mesh

Fig. 1. Illustration of 3D representations used as neural network input Volumetric grid (Section 3), Multiple-viewpoints renderings (Section 4), Point cloud (Section 5), and Mesh (included in Section 6). Example shape: ModelNet40 airplane_0627.

or LIDAR) which contain noise and occlusion. There are also works which include an overview of classification methods using various representations of CAD models by Su et al. [13], Shen [14] and Wang et al. [15]. However, each paper contains only a few selected approaches. Bronstein et al. [16] provide a comprehensive review of methods working with non-Euclidean input data – graphs and manifolds. Rostami et al. [17] and Ahmed et al. [18] survey and categorize approaches also suitable for CAD models classification but include only a few recent deep-learning-based approaches. Compared to [18], we provide a more fine-grained categorization of classification approaches derived independently of their work but similar to their Input-oriented taxonomy. Several works also compare the practical performance of various network architectures on a benchmark dataset – some only summarize the reported accuracies [9, 18, 19], while others replicate the performance evaluation and independently measure the accuracies [20, 21, 22].

1.2 Article Structure

The following sections 2 through 7 give an overview of methods using different shape representations and describe approaches for each representation in detail. In sections 8 and 9, we introduce our evaluation methodology and present the results of our experiments.

2 Survey of 3D Classification Methods

Shapes can be represented in various formats depending on the use-case or data acquisition method. In CAD applications, freeform surfaces or CSG (constructive solid geometry) models are commonly used [23, 24], providing precise information about object shape; other modeling software uses polygon mesh to represent the approximate shape of the object. In applications where the object is not created on a computer, its shape is measured in real world using suitable sensors. In medical applications, computer tomography or magnetic resonance imaging are used to produce volumetric scans, and in automotive and robotics industries, the object surface is scanned using an RGBD camera or a LIDAR producing point clouds. Some reviewed classification approaches work with the native representation for their data source, others convert the original representation to another format more suitable for processing with a neural network.

The usual approximative polygon or triangle mesh surface representation of 3D models in available datasets is non-regular as triangle sizes may differ within a model and triangle counts may differ between individual models, unlike e.g. images, where the resolution and pixel dimension are fixed. Therefore such representation is a challenging input to be directly processed by a neural network commonly working with regularly structured input and only a few networks use it as an input representation. Conversion to a different representation is often used to pre-process the mesh to a more suitable format.

2.1 Categorization of Approaches

We provide a hierarchical categorization of the surveyed approaches based on the following criteria. First, we classify the networks according to the shape representation they use as their input: volumetric grid-based, multiple-viewpoint image-based, point cloud-based, networks which process the object's shape or mesh approximation, and hybrid methods that process multiple representations simultaneously. Basic representation types are illustrated in Figure 1. Second, within each category, we couple the surveyed methods into categories by the similarity of the used algorithms and approach ideas and describe essential properties of each method's architecture. The resulting categorization is presented in Table 1. Figure 2 shows the overview of reported accuracies.

Classification neural network architectures can be divided into two parts: a *feature extractor*, which transforms the input shape representation to a feature vector, called descriptor, and a *classifier*, which learns to transform the extracted features into scores denoting the probability of individual classes. The feature extractor differs based on the input representation and is usually designed based on each approach's unique ideas. Depending on the approach, these parts may be trained together (*end-to-end*) or separately. The following sections describe each representation in more detail.

3 Volumetric Grid-based Neural Networks

In volumetric convolutional neural networks (CNNs), the convolution operation is used for the feature extraction task. It exploits the spatial locality of low-level features such as edges and can be applied hierarchically, usually with pooling or striding, reducing the resolution and increasing the level of abstraction in each step – similarly to CNNs that process 2D images. As 2D convolutional neural networks were a significant breakthrough in image classification [5], it is natural to generalize this approach to three dimensions. Instead of pixels, a 3D occupancy grid of *volume elements* or *voxels* is used. However, the 3D convolution operation is computationally more demanding, and volumetric grids

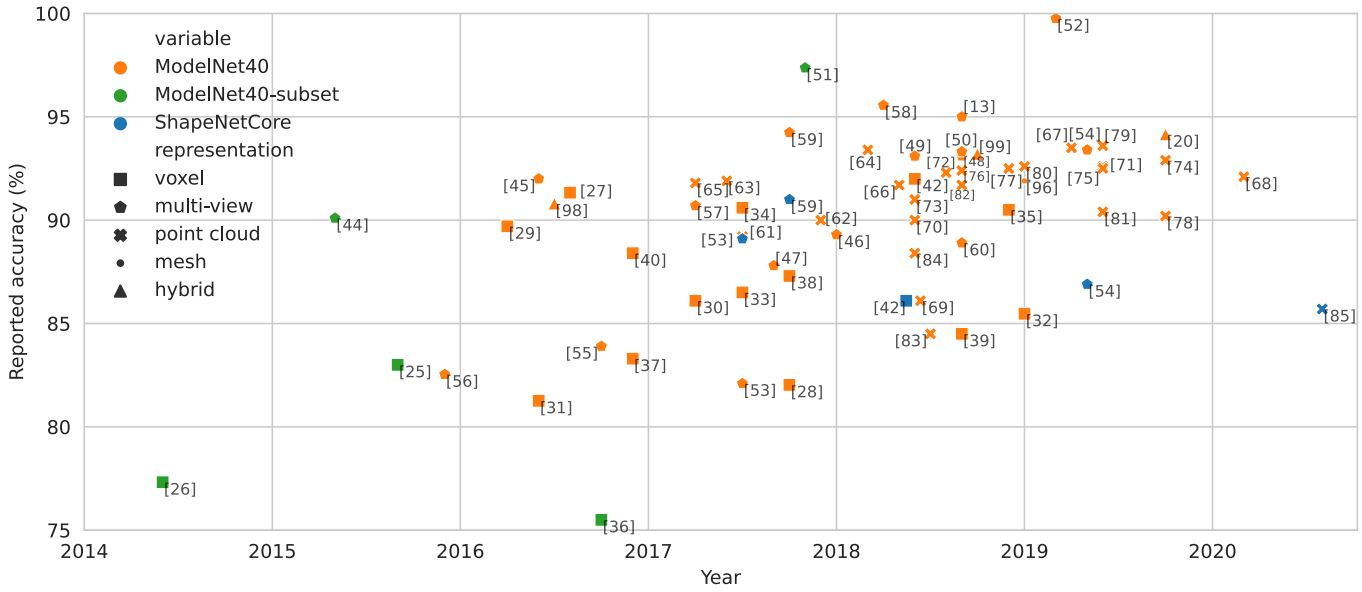


Fig. 2. Reported accuracies of the surveyed methods over time. Datasets and input representations are denoted by different colors and shapes.

TABLE 1
Taxonomy of the surveyed approaches. The references in bold were included in our evaluation

Volumetric grid	Basic Architectures		[25] [26]
	Voxel CNN with Residual Connections		**[27]** [28]
	Auxiliary Task		[29]
	Network Architecture Optimization		[30] [31] [32]
	Octree-represented Voxel Grid		[33] **[34]** **[35]**
	Unsupervised Representation Learning		[36] [37] [38] [39]
	Non-convolutional Approaches		[40]
	Conversion from a Point Cloud		[25] [41] [42]
Multi-view (images)	Basic Architectures		[43] [44] **[13]**
	Multiple Modalities		[45] [46]
	Axis-aligned Views		[47] [48]
	Learned View Grouping		[49] **[50]**
	Unsupervised Viewpoints Assignment		**[51]** [52]
	Unsupervised Representation Learning		[53]
	Using Auxiliary Data		[54]
	Special Projections	Geometry Images	[55]
		Panorama	[56] [57] [58] [59]
		Spherical	[60]
Point cloud	Symmetric Operation on Points		**[61]** [62]
	Hierarchical Feature Extraction		**[63]** **[64]** **[65]**
	Convolution on Neighborhood Graph		[66] [67] [68]
	Convolution on Points	Grid Around the Query Point	[69] [70] [71]
		Continuous Convolution	[72] [73] [74] [75] [76] [77] [78] [79]
	Sequential Processing	Using Attention Mechanism	[80] [81]
		Encoding Locality into Order	[82]
	Unsupervised Learning of Shapes		[83] [84] [85]
Surface shape	Manifold-based Convolution		[86] [87] [88]
	Graph-based Convolution		[89] [90] [91] [92] [93]
	Native Mesh-based Approaches		[94] [95] [96] [97]
Hybrid	Ensembling		[98] [13] [20]
	Descriptor Merging		[99]

tend to have high memory requirements as the voxel count grows with the cube of the spatial resolution. For this reason, only relatively low-resolution grids can be used, the most usual being 32^3.

Each voxel contains a value representing the presence of the object at a given place: several networks [25, 26, 31, 33] use a binary occupancy grid where each voxel is assumed to be either entirely occupied by the object (inside it) or empty (outside of the object). This format can be used for voxelized watertight 3D meshes as the "is inside" predicate is well-defined. Another voxelization option is to occupy only the voxels intersected by the object's boundary (both voxels inside and outside of the object are empty), typical for voxel grids converted from point clouds. Non-binary values may be used for encoding the point cloud density [41], but the difference in results is negligible [29]. Additional input channels such as surface normals may also be encoded in each grid cell [34].

3.1 Basic Architectures

VoxNet [25] is the first of the successful systems applying 3D convolutions to occupancy grids for classification, which we use as an example of a network with a convolutional architecture. In VoxNet, the occupancy grid is processed by two 3D convolutional layers, which extract local features and lower the resolution. The convolution result is passed to a leaky ReLU layer to achieve nonlinearity. Maximum pooling is then performed to get better representation and further lower the number of parameters needed. Finally, the resulting 3D feature map is flattened and passed through two fully connected layers, which output a vector with category probabilities. Figure 3 shows a diagram of the VoxNet architecture.

As is typical with neural networks, data augmentation is a crucial part of the training process. VoxNet uses rotation along the vertical axis as its main augmentation technique. It uses n copies of each input instance during training, each

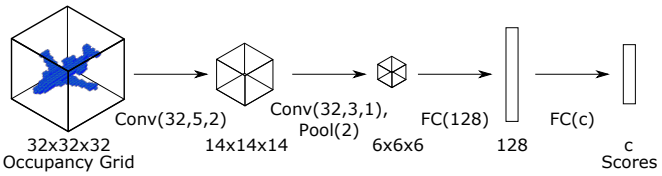


Fig. 3. VoxNet [25] architecture. c is the number of output categories.

rotated by $360/n$ degrees – typical values of n range from 8 to 24. At evaluation time it presents all rotations of the input object to the network and then uses pooling across the rotations to get the class prediction. The authors reported classification accuracy of 83% on a subset of the ModelNet40 dataset [26], which we briefly describe in Section 8.1. An official VoxNet implementation in Theano+Lasagne is available [100].

A similar 3D convolutional architecture is presented in **3D ShapeNets** [26]. It consists of 3 convolutional layers followed by a two-layer Restricted Boltzmann Machine classifier. Apart from presenting a classification network architecture, the authors also publish the ModelNet dataset. The reported classification accuracy on the ModelNet40 subset is 77.32%. An official implementation in Matlab is available [101].

3.2 Voxel CNN with Residual Connections

Voxception-ResNet [27] is inspired by deep residual convolutional networks for image classification, which are the state-of-the-art approach for this task. It uses batch normalization [102] and residual connections [7]. The network consists of several sequential *Voxception* (Inception-style [103]) modules allowing stochastic network depth [104] between 8 and 45 layers, which should enable information to propagate in the network through many possible "pathways". The best-performing architecture consists of Voxception blocks and downsampling blocks, enabling the residual network to choose the best downsampling methods (e.g., convolutions with stride greater than one or pooling). Voxel grid resolution of 32^3 is used. The network is trained using 24 rotations of each input instance along the vertical axis and using occupancy grid with values $\{-1, 5\}$ instead of $\{0, 1\}$ to encourage the network to pay more attention to positive entries. Voxception-ResNet (VRN) architecture achieves 88.98% accuracy on ModelNet40 dataset for single voxelization *view* and 91.33% with 24 *views*. An ensemble of similar models achieves 95.54%, which remains one of the highest reported for voxel-based networks. The authors have published their implementation in Theano with Lasagne [105].

Arvind et al. [28] explore the impact of residual layer width in volumetric networks on the classification accuracy. They show that increasing the number of channels in 3D convolutional layers improves the classification accuracy. This result extends the previous observations on residual 2D convolutional networks to 3D CNNs. They use a shallower architecture consisting of one classical 3D convolution, max-pooling and two residual-convolutional blocks, each followed by two *identity blocks* (two single-voxel convolutions) acting as a two-layer perceptron with shared weights across spatial dimensions, average pooling and a final classification layer. The best single-network model found by hyperparameter grid search achieved 82.03% classification accuracy on ModelNet40, or 86.50% by ensembling 10 independent training sessions. As of writing, there is no implementation available.

3.3 Auxiliary Task

Using an auxiliary task can improve classification accuracy. In the context of 3D CNNs, this has been shown in **ORION** [29] by simultaneously training the network for pose estimation. Apart from the object class, the network outputs rotation around the vertical axis. As different object categories have different rotational symmetries, the authors estimate the orientation by classifying the canonical rotation, where the number of orientation classes is dependent on the object category. This network achieves accuracy of 89.7% on the ModelNet40 dataset with manual rotation alignment. An implementation in Caffe and a manually-aligned version of the ModelNet40 dataset are available [106].

3.4 Network Architecture Optimization

As the voxel representation and 3D CNNs are computationally demanding, there have been attempts to speed up the training and inference, such as manually or automatically optimizing the number of layers or their hyper-parameters or replacing floating-point weights with binary values.

LightNet [30] reduces the size of the VoxNet model by adding one more convolutional and max-pooling layer, reducing the resolution passed to the fully connected part of the network to $2 \times 2 \times 2$ with 128 feature channels. LightNet also uses multi-task learning: *subvolume supervision* and orientation estimation. Therefore there are two fully connected classifier branches: main and auxiliary, both outputting the category and 12-class orientation prediction. The main branch uses all the extracted features by the convolutional part, and consists of a two-layer perceptron with a final softmax activation. The auxiliary branch consists of eight fully connected layers with softmax activation, each taking 128 inputs – each from one voxel in the $2 \times 2 \times 2$ feature map – and outputting both the category and orientation prediction independently on the other parts of the auxiliary branch. This forces the convolutional part of the network to extract meaningful features so that the class and orientation can be determined even from parts of the input shape. The final model has only 0.3M trainable parameters, less than a third of the original VoxNet architecture, while achieving superior ModelNet40 classification accuracy of 86.1%. The code was not publicly available as of writing.

Xu and Todorovic [31] describe a process of automatic CNN architecture optimization by iterative layer or filter addition while keeping the trained weights. The best-performing architecture found by this approach consists of 3 convolutional layers followed by one fully connected (FC) layer. Compared to 3DShapeNets [26] the authors achieve better accuracy of 81.26% with less than 1% of the trainable parameters. An official implementation in Matlab is available [107].

Ma et al. [32] show the reduction of memory footprint and computational demands by converting layers inputs and

weights to binary values, adding necessary batch normalization layers and sign activation functions. Although in most cases this process decreases the classification accuracy, in the case of **binVoxNetPlus**, a network constructed by adding one additional convolutional and a fully connected layer to VoxNet, the accuracy is increased by binarization from 83.91% to 85.47% while the model size was just 0.29 MB, approximately 8% of the original VoxNet. There was no public implementation at the time of writing.

3.5 Octree-represented Voxel Grid

As memory and computational demands limit the resolution of the voxel grid, a more efficient representation is needed to make processing a higher-resolution voxel grid feasible. Voxel occupancy grids representing real-world objects or their surface are usually sparse; therefore, large regions without changes in occupancy do not need to be stored or processed in full resolution.

Octree [108] is a tree where each internal node has exactly eight children, partitioning the space into finer and finer cubes ($2 \times 2 \times 2$) until a depth limit is reached or the space corresponding to a node is homogeneous. This allows efficient voxelization of the 3D model where only voxels on the object boundaries are stored at high resolution. Convolution and max-pooling can be adapted to work on octrees. The octree structure can be processed efficiently on GPUs when stored suitably, allowing fast training and inference.

OctNet [33] uses octree to efficiently represent an occupancy grid, implementing convolution and pooling operations directly on octrees. This reduces the memory and computational requirements, allowing one of the presented models to process occupancy grids of 256^3 resolution at memory requirements lower than if the model used the classical voxel representation at 64^3 resolution. The proposed network architecture consists of repeated blocks of two 3^3 convolutional layers and one max-pooling layer, halving the resolution until a 8^3 feature map is reached, which is then passed to a fully connected classifier. The classification performance is on par with similar networks working with a dense voxel grid, achieving ModelNet40 classification accuracy of approximately 86.5% with 128^3 input resolution. As the authors focused on measuring the impact of increasing the resolution, minimizing impact of other factors, a "pure" 3D CNN is proposed, unused methods such as data augmentation are likely to further improve the performance. An official implementation in Torch is available [109].

O-CNN [34] uses octree to store the object surface, with additional information: normal vector of the surface stored in leaf nodes. The authors present an efficient convolution operation on octrees and construct a hierarchical structure of shared layers for individual levels of the tree. Global feature computation proceeds from the finest leaf octants and continues upwards to the root of the tree. The high-level O-CNN architecture used for classification consists of repeated 3D convolutions and max-pooling followed by two fully connected layers with dropout and a final softmax activation. The O-CNN with five convolutional layers processing the input with resolution of 64^3 achieved 89.9% in ModelNet40 classification accuracy for single input and 90.6% with 12-rotation voting. Standard octrees have a fixed maximum depth, wasting memory on flat regions where a simple planar approximation would be sufficient. Adaptive Octree representation used in **Adaptive O-CNN** [35] uses planar patches as a representation in leaf nodes. Flat areas of the original mesh can be represented by a single leaf on a higher level of the tree, while more complex areas are subdivided into finer details. Both computation time and memory requirements are reduced compared to the O-CNN network, in case of the 256^3 input resolution, where the change is the most significant, to approximately $1/4$. Apart from the convolutional encoder for feature extraction, a decoder is also proposed, allowing other tasks such as 3D autoencoding and shape completion. The achieved ModelNet40 classification accuracy is 90.5% with 12-orientation voting with input resolution of 32^3. The authors offer an implementation of both networks in Caffe and tools for converting mesh data to octrees and adaptive octrees [110].

3.6 Unsupervised Representation Learning

All approaches described so far learn features for classification in a supervised training scenario – providing the correct category labels to the network when running back-propagation training pass. This section describes approaches with an additional training step – unsupervised learning of features on a different task such as reconstruction of the input, using an auto-encoder, to force the network to find a meaningful latent space or "bottleneck" representation. The input-to-latent-space mapping learned by the encoder can be used for feature extraction. In the next step, a classifier can be trained to transform these features into category labels of the input object in a supervised fashion. The classifier can be either a separate model (e.g. SVM or a neural network), or the decoder can be replaced with e.g. a fully connected classifier, and the network can be trained end-to-end, fine-tuning the previously found input-to-latent-space mapping.

VConv-DAE [36] is the first attempt to learn a shape embedding in an unsupervised way. The authors use a denoising autoencoder with two convolutional (and de-convolutional) layers that aims to learn a 6912-dimensional latent representation of training shapes. After training the autoencoder on a subset of ModelNet dataset the authors compare following two approaches for classification: using an SVM classifier, or attaching two fully connected layers after the encoder and fine-tuning the network end-to-end. The former approach achieves 75.50% ModelNet40[2] classification accuracy, proving the network found some category-identifying features on its own. The fine-tuned model achieved 79.84%. An official implementation in Torch is available [111].

3D-GAN [37] is a fully 3D-convolutional generative adversarial network [112] with a 5 convolutional layers generator and discriminator trained on voxelized models from the ShapeNetCore dataset. Concatenation of several discriminator layers' outputs (max-pooled 2., 3. and 4.) is passed to linear SVM for classification. This model achieves 83.3% classification accuracy on the ModelNet40 dataset. The authors provide an implementation in Torch [113].

2. Using a ModelNet40 subset with equal category model counts and therefore different train/test split.

Wang et al. [38] present **3D-ED-GAN**, a combination of 3+3 convolutional layers encoder-decoder with adversarial loss (discriminator) trained to reconstruct damaged, lower-resolution input. Although the authors focus mainly on inpainting, the encoder part of the network is used for classification in one of the presented experiments. When using only the encoder to extract features, passing them to a linear SVM, the ModelNet40 classification accuracy of 84.3% is achieved. The ModelNet40 classification accuracy of 87.3% is achieved with pre-training the encoder-decoder on ShapeNetCore in an unsupervised fashion, then using the latent representation as input of a classifier with softmax activation. When the same model was trained from scratch (without pre-training, with randomly initialized weights) the accuracy of 86.1% was achieved, 1.2 pp lower than the fine-tuned result. As of writing, only an unofficial[3] implementation in TensorFlow is available [114].

Variational Shape Learner [39] is a variational encoder-decoder learning the latent representations of shapes in an unsupervised fashion and using the representation for classification. Apart from producing just one global feature vector (latent code) by passing the input voxel occupancy grid through three convolutional layers followed by two FC layers, local latent codes are also computed. Each local feature is computed by two FC layers, whose input is the concatenation of a sample of the global feature vector and the previous local latent code – the output of the previous *local* layer, if any. The authors argue this approach helps the network better learn a hierarchical representation of input shapes, where each local feature vector represents a higher level of abstraction. This approach achieves 84.5% accuracy on ModelNet40 by training an SVM on extracted features – concatenated global and local latent features. An official TensorFlow source code is available [115].

3.7 Non-convolutional Approaches

FPNN [40] presents a lightweight, non-rigid alternative to convolution. Compared to convolution, the weights at given positions are optimized together with the positions of the probing points. First the voxel occupancy grid is converted to a 3D distance field – a voxel grid containing values representing a continuous distance from the surface; distance at arbitrary position is computed as a tri-linear approximation using nearest cells. The distance field is "sensed" with probing points (filters), whose weights and positions are learned. The distances at probed positions and the trained weights are combined by dot product. Similarly to the object shape being represented with a distance field, normals are stored in a three-channel normal field. The proposed non-hierarchical architecture passes the distance and normal fields through field probing layers, producing an internal representation of the input shape, from which the object class is extracted with fully connected layers. This approach achieves 88.4% ModelNet40 classification accuracy with 4 fully connected layers at the end, with distance field and normals on input, or 87.5% without the normal field. An official implementation in Caffe is available [116].

3. Differs from the paper: needs modification to extract the latent representation for classification, last decoder layer has 512 filters, not 256 as in the paper, possibly other differences.

3.8 Conversion from a Point Cloud

Point cloud processing with a convolutional neural network is not trivial, as the standard convolution requires its input to be a regular grid. There are approaches which take a point cloud as input, convert it to a voxel grid, and apply 3D convolutions for feature extraction followed by classification.

VoxNet, which was mentioned earlier, uses a voxel occupancy grid generated from a point cloud input, passing the voxel grid to the CNN's input.

PointNet by Garcia-Garcia et al. [41][4] integrates the point cloud to occupancy grid conversion into the network as a non-trainable layer. The grid representation is then processed like in other voxel-based networks by repeated 3D convolutions and pooling followed by a 3-layer perceptron.

PointGrid [42] achieves input regularity by sampling a predefined number of points from the input point cloud in each cell of a regular 3D occupancy grid, concatenating relative coordinates of the sampled points in each voxel, so they can be used as features of the voxel. The rest of the model consists of 9 3D convolutions and max-pooling followed by FC layers. This architecture achieves 92.0% ModelNet40 classification accuracy and 86.1% accuracy on the ShapeNetCore dataset. The official source code written in TensorFlow is available [117].

4 Multiple-viewpoint Image-based Neural Networks

Multi-view image-based networks use 2D images of the object as input, captured or rendered from different viewpoints. A general setup of a multi-view classification network is as follows: the images (views of a single 3D shape) are passed to one or more feature extractors, then a technique for combining features from different views is employed, and the resulting feature vector is sent to a classifier. Feature combining techniques range from simple pooling across the views to employing a recurrent neural network to process them as a sequence. Being based on image feature extractors allows reusing existing image-processing CNNs, enabling transfer learning and leading to good results.

The individual multi-view approaches generally differ in several aspects: the number of viewpoints and their spatial distribution, input modality, whether they use multiple modalities generated from each viewpoint, which 2D CNN architecture they use for image feature extraction, whether the weights are shared among viewpoints, and in the way, they aggregate the features extracted from the rendered images.

4.1 Basic Architectures

The first *multi-view* CNN approach for 3D object classification by Lecun and Huang [43] uses two views as input (simulating binocular vision), which are sent to a shallow convolutional neural network. The architecture consists of three convolutional layers with intermediary sub-sampling layers, followed by one fully connected layer, which predicts category probability vector. Half of the convolutional filters in the first layer are applied to only one of the two images

4. Not to be confused with the PointNet by Qi et al. [61].

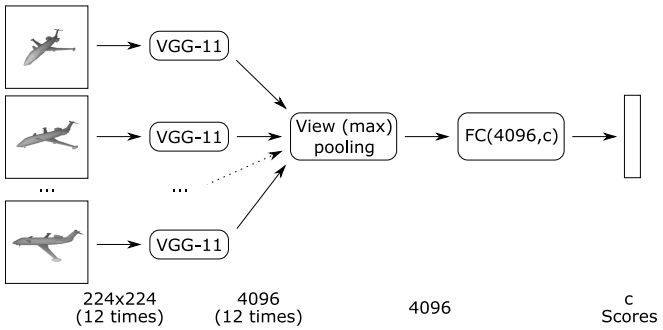


Fig. 4. Multi-view architecture [44], as used in [13]. The numbers below the diagram denote tensor sizes; c is the number of output categories.

(left or right), the other half process both. All features are then merged in the last convolutional layer. As this work predates the ModelNet dataset, it has only been evaluated on a dataset of physical uniform-colored toys photos with varying viewpoints and lighting, released with the paper.

The first approach to process more than two views for classification is **MVCNN** [44]. It uses feature extractors based on VGG [6], sharing weights for individual viewpoint branches, then applies max pooling across the views to combine the extracted features. Resulting feature vector is passed to three sequential fully connected layers for classification. When training, the network is first trained as a single-view classification network (single feature extraction branch, without max pooling), then other view branches are added and the network weights are fine-tuned. The authors use two different viewpoint configurations: either 12 "turntable" views[5] or 80 virtual cameras placed in 20 vertices of a dodecahedron around the object, each with 4 rotations in 90 degree increments to detect objects with non-upright pose. The 12-view network achieves ModelNet40[6] classification accuracy of 89.9% with pre-training on ImageNet, and the 80-view variant achieves 90.1%. The authors published an implementation in Matlab [118].

A revisited but similar approach is presented by Su et al. [13]. The authors explore different pre-trained image network architectures and image rendering techniques, significantly improving accuracy of this method. The best-performing classification model takes 12 Phong-shaded images as input uses a pre-trained ResNet-34 convolutional network as its base, and achieves 95.9% accuracy on the ModelNet40 benchmark. When the VGG network is used for feature extraction, the accuracy of 95.0% is achieved when fine-tuning its weights initially trained on ImageNet1K, and 91.3% without pre-training. The authors also tried using depth images concatenated to the shaded images, which yields 96.2% classification accuracy. An illustration of the multi-view architecture is shown in Figure 4. The authors offer a publicly available PyTorch implementation[119], utilizing the VGG CNN as the feature extractor.

5. Twelve virtual cameras regularly distributed at 30-degree elevation from the object centroid.

6. Limited to 100 shapes per category; published dataset: 3983 shapes.

4.2 Multiple Modalities

The classification accuracy may benefit from extra information in addition to the shaded images, such as depth or surface curvature, passed as additional images to separately-trained feature extraction branches of the network. There are more approaches based on this idea.

Johns et al. [45] process a pair of images in two modalities (shaded and depth) from multiple viewpoints. The images are processed with a CNN to extract features passed to a 3-layer fully connected network to predict a class label and next-best-view viewpoints. This model achieves ModelNet40 classification accuracy of 89.5% (6 viewpoints chosen by the network) or 92.0% (12 views). No code was available at the time of writing.

The classification approach described by Minto et al. [46] uses axis-aligned images with multiple rendered modalities: depth, voxel density (X-ray-like image), and estimated surface curvature constructed by fitting a parametric surface (NURBS) to the points constructed from depth-image samples. The images are passed to a 4- or 5-layer CNN, whose weights are shared among branches processing the images of the same modality, which are rendered from viewpoints rotated around the vertical axis. The extracted features are then concatenated and passed to a fully connected layer with a softmax activation for classification. This achieves 89.3% classification accuracy on ModelNet40. No code was available at the time of writing.

4.3 Axis-aligned Views

Some approaches restrict the camera positions to only axis-aligned viewpoints, while using one modality.

Zanuttigh and Minto [47] use six axis-aligned depth images rendered from coordinate axis directions, looking toward the origin. The images are passed to CNNs consisting of four 7×7 convolutions, in a branch for each view, sharing weights for rotations around the vertical axis. The feature extractors' outputs are concatenated and sent to a single-layer classifier. This approach yields ModelNet40 classification accuracy of 87.8%. No code was available at the time of writing.

Sarkar et al. [48] propose to use multi-layer heightmaps as an input image modality. Multi-layer heightmaps are depth images rendered from multiple planar slices through the object, in this case, axis-aligned. Three views with a heightmap of each slice, stored as individual channels, are passed to VGG16 (with batch normalization, without FC layers), the resulting features are then merged by concatenation along the depth or channel dimension to preserve the information about which feature comes from which view. A convolutional layer then reduces the channel count, and the features are passed to one FC layer for classification. This model achieves 93.11% classification accuracy on ModelNet40 with 5-layer heightmaps and VGG CNN pretrained on ImageNet. The authors have published their PyTorch code [120].

4.4 Learned View Grouping

Rather than using simple max- or average-pooling to aggregate features from multiple viewpoints, these approaches use a trainable model, promising to learn to use the most important features from each view.

GVCNN [49] adds a view-grouping step between the extracted features and the classifier, considering the correlation among individual views. The features are extracted by passing each view (shaded image) through the first five convolutional layers of GoogLeNet[7] independently. Then per-view discrimination scores are computed by pooling the extracted features, and views are assigned to groups based on their discrimination scores – quantized to bins expressing "how well does this view perform in classification". Then the view groups are pooled using a grouping scheme, which is selected based on the final group count. On ModelNet40 this approach achieves 93.1% accuracy with 8 views, pre-trained on ImageNet1K. The authors have not published the code, but there are two unofficial implementations: in TensorFlow [121] and in PyTorch [122].

SeqViews2SeqLabels [50] employs a recurrent neural network and treats multiple views as a sequence of images using an encoder-decoder architecture. The network outputs a sequence of class labels rather than probability vectors. A pre-trained VGG-19 network, fine-tuned on the single-image classification task, is used as per-view feature extractor. Feature vectors are fed into the encoder as a sequence. Both the encoder and decoder consist of GRU cells [123]; attention is used in the decoder for selecting distinctive views for current output label. This architecture achieves 93.31% classification accuracy on ModelNet40. An implementation in TensorFlow is available [124].

4.5 Unsupervised Viewpoints Assignment

Unlike other multi-view networks, these networks do not obtain information about the position of the viewpoints, i.e., the viewpoints can be rotated arbitrarily. It is the task of the network to find a suitable assignment of input images to viewpoints.

RotationNet [51] combines the multi-view classification with an unsupervised pose estimation task. This encourages the network to train one CNN per viewpoint to classify the object in an input image from a specific viewpoint. A new category is added to the set of original classification categories to assess the image-to-viewpoint assignment's correctness. The meaning of this category is "this is not the correct viewpoint". Such per-viewpoint training helps the CNNs by reducing the input image variations, allowing the trained convolutional filters to be more view-specific. The image-to-viewpoint assignment is trained in an unsupervised fashion, trying multiple assignments during inference, choosing the most probable one[8] according to predicted categories. The authors report maximum accuracy of 97.37% on the same subset of ModelNet40 dataset as used in [44]. The authors published two implementations, one in Caffe [125], and another one in PyTorch [126].

Sarkar et al. [52] present a RotationNet-like network trained on images produced by rendering slices of the input model. They render multi-layer heightmaps or occupancy slices images – a binary image representing intersections of the model with the slicing plane. The authors report the accuracy on ModelNet40 of 99.76% when using 12 turntable views in occupancy slices modality with 10 slices. The code had not been published at the time of writing.

7. GoogLeNet = Inception_v1, but Inception_v4 is used in both available unofficial implementations

8. The one with the least probability of being in the "incorrect viewpoint" category

4.6 Unsupervised Representation Learning

Like in the case of volumetric networks, the classification can be achieved by first training an auto-encoder network in an unsupervised fashion, and then using the bottleneck representation as extracted features, only appending a classifier.

Soltani et al. [53] use unsupervised representation learning working on images of one of multiple modalities. The authors use a variational autoencoder built from four residual convolution blocks in the encoder, working with multi-view depth maps or silhouettes rendered from 20 viewpoints in dodecahedron vertices. While not focusing on reaching state-of-the-art classification accuracy, the proposed **AllVP-Net** architecture with two-layer dense classifier appended to the encoder achieves ModelNet40 classification accuracy of 82.1% and 89.1% on their split of ShapeNetCore, both with depth images used as input. No fine-tuning was used. The authors offer an implementation in Torch [127].

4.7 Using Auxiliary Data

Some 3D shape datasets provide part segmentation information useful to aid further the classification process.

Parts4Feature [54] proposes using local part detection by extracting features from each view and using a region proposal network [128] to produce a representation of possible parts visible in each view. For classification, top k proposed regions are aggregated by an attention mechanism and part co-occurrence patterns learning. Using part segmentation information during training, the model achieves 93.40% accuracy on ModelNet40 and 86.9% on the ShapeNetCore dataset. No code was available at the time of writing.

4.8 Special Projections

There are also approaches which use other projections than perspective or planar projections from multiple viewpoints, described earlier.

4.8.1 Geometry Images

A 2-manifold 3D mesh can be represented as a so-called geometry image by cutting the mesh along edges and mapping the unwrapped mesh into a square image domain containing channels representing the original coordinates and local properties such as curvature. Sinha et al. [55] take a geometry image of 56×56 pixels as an input with the following channels: two principal curvatures, topological mask and height field. The proposed CNN extracts a 96-dimensional feature vector from the image, which is passed to a fully connected layer for classification. The approach achieves ModelNet40 classification accuracy of 83.9%. The code for generating geometry images has been published [129], but we have not found the code for the CNN or its training.

4.8.2 Panorama Projection

Another projection of a 3D shape suitable for CNNs is a panoramic view. **DeepPano** [56] uses a single view of the 3D shape: a depth image created by cylindrical projection around the vertical axis, passed to a 4-layer CNN, row-wise max-pooling for rotation invariance, and two fully connected layers and softmax-activated output. This model achieves 82.54% classification accuracy on ModelNet40. The authors have published their Matlab code for panorama projection image rendering [130], but not the CNN implementation.

Another approach by Sfikas et al. [57] uses three views: rendering each model in cylindrical projection along each axis. Depth and normals' deviation map modalities are stored as image channels. Image features are extracted with a 3-convolutional-layer CNN and classified with one fully connected layer, dropout layer and softmax activation. The object pose is normalized before rendering. The classification accuracy of 90.70% is achieved on the ModelNet40 dataset. No public implementation is available.

In a follow-up paper, Sfikas et al. [58] also use three views: cylindrical projections corresponding to major axes with an additional modality: the magnitude of the normals' deviation map gradient. The architecture is identical to the previous model, except it has two fully connected layers instead of one. The authors achieve ModelNet40 classification accuracy of 95.56%. No code was available at the time of writing.

Cao et al. [59] use 12 views: vertical stripe-projections around the sphere (parallel to longitudes) and one horizontal stripe (cylindrical projection). The vertical slice images are fed into a MVCNN-like network with AlexNet feature extraction; similarly, a feature vector is extracted from the horizontal stripe. Finally, the two feature vectors are concatenated and passed to a fully connected layer with softmax activation for classification. Using depth and contours modalities the network achieves 94.24% classification accuracy on ModelNet40 and 91.00% on ShapeNetCore, both fine-tuning AlexNet weights pre-trained on the ImageNet dataset. No code was available at the time of writing.

4.8.3 Spherical Representation

The input shape can also be projected into a spherical domain and passed to a CNN with operations working in that domain. **Spherical CNN** [60] computes a spherical function similar to a depth map – distance to the first intersection of a ray cast from a point on an enclosing sphere toward the origin – and surface normals at the intersection points. Convolutions in the spherical harmonic domain are applied, and spectral pooling is used for rotation invariance. The architecture consists of two branches, one for each modality, each with 8 spherical-convolutional layers. The extracted features are then pooled, concatenated and passed to a classifier. This achieves ModelNet40 classification accuracy of 88.9% when only z-rotation invariance is required, and 86.9% for full SO(3) rotation invariance. The authors published an implementation in TensorFlow and code to convert voxel occupancy grids to spherical domain [131], but no code for mesh projection.

5 Point-cloud-based Neural Networks

A point cloud is a set of points in Euclidean space representing the surface of the object. It is a natural output format of laser scanning devices used by robots or autonomous cars. It can also be easily constructed from an artificially-modeled object by sampling its surface, e.g. mesh faces. Compared to volumetric grids or images, point clouds are neither structured nor ordered, which poses a challenge to neural networks: point-based operations need to be defined for successful feature extraction. Unlike **VoxNet** [25] and **PointNet** by Garcia-Garcia et al. [41] presented above, which convert the input point cloud to a dense volumetric grid and apply 3D convolutions, this section describes architectures which work directly with a set of point coordinates or a derived representation maintaining its unstructured and non-dense nature.

The approaches working with this representation differ the most in the feature extraction methods. Some process each point individually and apply a symmetric aggregation operation to achieve order-invariance. Others construct a nearest-neighbor graph and extract features from that, or define a convolution-like operation working with points by generalizing the input and kernel to non-rigid positions.

5.1 Symmetric Operation on Points

One of the first networks to successfully overcome the difficulties of processing raw unordered point clouds is **PointNet** by Qi et al. [61]. Its main idea lies in using only symmetric functions, i.e., functions for which the order of arguments does not matter. Each point is processed independently by a series of multi-layer perceptrons (MLP) sharing their weights. Then a global feature vector is constructed using max-pooling, which is a symmetric function. Another important feature of PointNet is the set of learnable geometric transformations, which ensure some invariance to rotation or jittering (random small translations) of the input point cloud. Rotation and jittering are also used as data augmentation during the training. Figure 5 shows the diagram of the PointNet architecture. The authors report ModelNet40 classification accuracy of $89.2\,\%$ on 1024 points. A TensorFlow implementation of this network is available [132]. The PointNet architecture (without T-Net blocks) is also implemented in Minkowski Engine [133], a generic machine learning and auto-differentiation framework supporting sparse tensors, which allows training and evaluation using different number of points in each object.

Deep Sets [62] is a concurrent research based on the same idea of using a symmetric function for points aggregation. In the proposed classification network, each input point is transformed independently into a learned internal representation by three dense layers with $\tanh$ non-linearity. Then max-pooling is used, which is a symmetric operation as in PointNet. The resulting vector is processed with two dense layers for classification. This model achieves ModelNet40 classification accuracy of $90.0\,\% \pm 0.3\,\%$ for 5000 points or $87\,\% \pm 1\,\%$ for 1000 points. An official implementation in PyTorch is available [134].

Although PointNet and Deep Sets achieve better classification accuracy than previous methods based on manually designed feature extractors, the performance was later

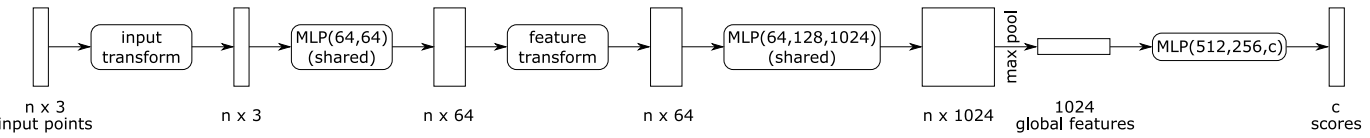


Fig. 5. PointNet [61] architecture. n denotes the number of input points, c is the number of classes.

improved by approaches which learn to aggregate local features.

5.2 Hierarchical Feature Extraction

The hierarchical structure of successful deep convolutional neural networks for image processing allows the networks to increase the level of abstraction at each layer. Several approaches apply similar hierarchical feature aggregation to point cloud feature extraction.

PointNet++ [63] presents a hierarchical structure inspired by convolutional neural networks which solves the problem of extracting local features. It clusters close sets of points together and runs the original PointNet on such neighborhoods. For this purpose, iterative farthest point sampling and multi-resolution grouping (which ensures good representation for regions with different densities) are used – fulfilling the same role as the pooling layer. Local features obtained in this way are represented by the centroid of the original neighborhood and clustered again in a hierarchical manner. Finally, a fully connected layer is employed to extract global features and perform classification. Several rotations of the 3D model are used during both training and evaluation to achieve better results. PointNet++ achieves ModelNet40 classification accuracy of 90.7% for 1024-point input or 91.9% for 5000 points including normal vectors as extra features in addition to positions. An implementation in TensorFlow is available [135].

The PointNet++ architecture lacks the ability to reveal the spatial distribution of the input point cloud during the hierarchical feature extraction. **SO-Net** [64] solves this problem by constructing a self-organizing map (SOM) which represents the point cloud better than simple centroids used in PointNet++. Each point of the original point cloud is associated with k nearest SOM nodes and for each such node a mini point cloud is constructed. This also ensures that the mini point clouds are overlapping, which is essential for improving the accuracy. These mini point clouds are processed by a series of fully connected layers similar to the original PointNet. This process yields a local feature vector for each of the original SOM nodes, which are then used for constructing a global feature vector by means of max pooling across the nodes. SO-Net achieves 93.4% classification accuracy on the ModelNet40 dataset. An implementation in PyTorch [136] is available, including the code for creating self-organizing maps from point clouds.

A kd-tree [137] is a data structure suitable for storing and searching in a set of points of higher dimension. Its 3D variant is used as an input format for **Kd-networks** [65]. First, a kd-tree is constructed over a point cloud. The tree is then fed to a series of layers in a recursive manner starting in the leaf nodes. The features are aggregated from child nodes at each level using a fully connected layer with ReLU nonlinearity, until the root node is reached. The global feature vector is extracted from the root and used for classification. Nodes on the same level of the tree, where the tree is split according the same coordinate, share the weights of the fully connected layers (e.g., weights are shared for all nodes at the third level of the tree, where it splits the points according to the x coordinate). During training, this network uses several geometric perturbations for data augmentation as well as randomized kd-tree construction. This approach can process raw point clouds but requires heavy preprocessing when constructing the kd-trees. This approach achieves 91.8% classification accuracy on the ModelNet40 dataset. The authors supply an implementation in Theano with Lasagne [138], also providing a framework for kd-tree construction from a point cloud.

5.3 Convolution on Neighborhood Graph

The input point cloud contains spatial information, which may be converted to another representation – a graph with nodes corresponding to the original points and edges connecting nearest neighbors, with original point positions stored as node features. Previously published graph-based convolution and pooling operations [89] can be used for hierarchical feature extraction at the expense of increased computational cost compared to previously described approaches.

Dominguez et al. [66] create a 6-nearest neighbors graph from a point cloud. Then graph convolution and pooling are applied repeatedly: a group of three graph-convolutional layers is followed by a pooling operation halving the number of points, all repeated four or six times for hierarchical feature extraction. A fully connected classifier follows this feature extractor. This approach achieves 91.7% classification accuracy on ModelNet40 for 516 points with pretraining on a subset of the full ModelNet dataset and voting over 10 augmented point cloud instances created by vertex dropouts and flips. A TensorFlow implementation of this network is available [139].

Dynamic graph CNN [67] proposes a convolution-like operation (EdgeConv) defined on edges of a k-nearest neighbor graph. The EdgeConv operation proceeds per-node: it applies a learned asymmetric edge function, implemented as a shared MLP, to each of k-nearest neighbor edges, and aggregates the results for given node neighborhood with a symmetric operation – max-pooling. The EdgeConv output is a higher-dimensional feature vector per each point. The proposed architecture transforms the input point cloud with a learned matrix multiplication to normalize it, and then passes the points through four sequential *EdgeConv* layers. The outputs of these layers are concatenated, pooled and classified with an MLP. Since the node positions are changed after each layer, the point cloud is not explicitly converted

to a graph representation – the graph is constructed "on demand" as needed from a set of points passed between layers. This approach achieves ModelNet40 classification accuracy of 92.9% for 1024-point input or 93.5% when sampled with 2048 points. Implementations in PyTorch and TensorFlow are available [140].

Lei et al. [68] propose efficient graph convolution and pooling operations on neighborhood graphs. The convolution kernel uses binning in discretized spherical coordinates similar to [70] weights are selected based on the position relative to the query point. Graph coarsening using farthest point sampling (to select point set of desired size to remain in the graph) is used to construct a multi-resolution pyramid of graphs; between its layers the convolution kernels are applied. Similarly, the order of pyramid layers can be reversed for upsampling. A classification network consisting of three-level pyramid encoder, global feature extraction and a fully connected classifier achieves ModelNet40 classification accuracy of 91.4% on 2048 points and 92.1% on 10000 points. A TensorFlow implementation is available [141].

5.4 Defining Convolution on Points

Similar to volumetric grid-based neural networks drawing from image CNNs, there have been successful attempts to adapt the convolution operation to work with point clouds. As in these networks, the convolution operation computes local activation as an aggregated product of a learned kernel and the neighborhood of each query point in the input. However, unlike voxel networks, the point convolution cannot assume a regular input (e.g. uniform density of the point cloud), but may use a regular kernel and point binning when computing the activation. In addition to the convolution operation, the pooling and sub-sampling operations also need to be adapted to point clouds to allow hierarchical feature extraction.

5.4.1 "Grid" Around the Query Point

In the following approaches, the kernel is a spatial structure centered at the position of the query point. The input points are binned in a grid based on their relative position from the query point, and weights are learned for the grid cells.

Hua et al. [69] define a pointwise convolution operator, which is applied at each point position, which essentially voxelizes a local neighborhood. The pointwise convolution centers a kernel at the currently processed point and bins close points to a regular grid around current point, aggregating point features in each cell, then a regular 3D convolution is used. To reduce the dependency on the input points permutation, the authors propose sorting the points in a canonical ordering (Morton curve). In the proposed classification network, the intermediate results from all four convolutional layers are concatenated and passed through two FC layers for classification. This achieves ModelNet40 classification accuracy of 86.1%. An official implementation in TensorFlow is available [142].

The spatial structure does not need to be a regular grid. Xie et al. [70] propose a convolution kernel where the current point neighborhood in relative radial coordinate system is split into a predefined number of bins in each coordinate. E.g. there are 8 quadrant bins with trainable weights for 2 polar and 4 azimuthal splits and no radial split. Points in each bin are aggregated by sum-pooling and the resulting per-bin vectors are transformed to a feature vector of the desired dimensionality. The best classification model consists of 5 **ShapeContext** layers, max-pooling and a MLP classifier. This approach achieves 90.0% accuracy on ModelNet40 for 1024 points, 27-bin kernels (3 azimuthal $\times$ 3 polar $\times$ 3 radial, max. radius = 0.5) or 89.8% for a learning model, where the bins are not hand-crafted, but the points are processed with a self-attention block. There was no code available at the time of writing.

Another option outlined in **A-CNN** [71] is to compute a convolution of points in a ring around a point. The order of points is "normalized" by sorting them counter-clockwise when querying for point neighbors inside a ring after projection to an estimated tangent plane. The annular convolution is a 1D convolution of a $1 \times k$ kernel and the ordered neighbors in a ring. The authors argue that compared to multi-scale neighborhood extraction (e.g. in PointNet++) the proposed ring approach does not have overlaps of individual areas at different scales, allowing the weights to be optimized using more discriminative, less redundant data. The classification network consists of repeated farthest point subsampling and annular convolutions to increase the receptive field of the extracted features, max-pooling and MLP for classification. The trained model achieves ModelNet40 classification accuracy of 92.6%. Authors' implementation in TensorFlow is provided [143].

5.4.2 Continuous Convolution

Rather than defining a discrete spatial structure, the convolution operation can be naturally defined on a continuous domain. The following approaches outline some of the possible definitions.

Point Convolutional Neural Networks [72] define a convolution operation on point clouds by extending the function defined over the point cloud to continuous 3D space. Then the classical convolution operation is applied, and the result is converted back to a point cloud by sampling. The proposed classificaton network consists of 3 convolutional layers followed by two FC layers. The classification accuracy on ModelNet40 is 92.3% with 1024 points. An implementation in TensorFlow is available [144].

Shen et al. [73] propose a kernel correlation layer, a convolution-like operation on point sets producing activations based on the correspondence of the kernel and the query point neighborhood. The kernel is a set of points with learnable positions. The classification network model extracts the low-level features by kernel correlation for each point, concatenates them with input points' positions and passes the result through a series of MLPs with a concatenated graph max-pooling "side-branch" to aggregate local features of nearby points. The model achieves ModelNet40 classification accuracy of 91.0%. The authors' implementation in Caffe is available on request [145].

KPConv [74] also uses a kernel consisting of a set of points with learnable positions and also weights. The proposed 5-layer classification networks achieve 92.9% and 92.7% classification accuracy for rigid and deformable kernels, respectively. An official TensorFlow implementation is available [146].

PointConv [75] is another approximation of continuous convolution by a set of points. To approximate continuous weight functions, MLP is used and weights are compensated for non-uniform point density: $\frac{\text{features}*\text{weights}}{\text{density}}$. The classification architecture consists of a feature extractor with 4 pointwise-convolution layers and a 3-layer perceptron. The authors report ModelNet40 classification accuracy of 92.5% when using 1024 points with normals. An official PyTorch implementation is available [147].

SpiderCNN [76] also proposes a continuous kernel generalization. The learnable kernels are based on step function and Taylor polynomials – to reduce the number of parameters needed in contrast with MLP. The best-performing classification network consists of 4 SpiderConv layers, whose outputs are concatenated, top-2-pooled and passed to a MLP classifier. SpiderCNN achieves ModelNet40 classification accuracy of 92.4%. A TensorFlow implementation of the architecture was published by the authors [148].

PointCNN [77] introduces a $\mathcal{X}$-Conv operator – a method applying a typical convolution to point clouds by first weighting and permuting input features with a learned transformation of the point coordinates relative to the query point. The $\mathcal{X}$-transformation is based on an MLP, which learns the weights of individual points and their latent order. The classification network consists of two $\mathcal{X}$-Conv layers subsampled with dilation d: instead of passing k nearest neighbors to the $\mathcal{X}$-Conv, $k * d$ nearest neighbors are found and k points are sampled from them. The $\mathcal{X}$-Conv layers are followed by four FC layer branches, each using different outputs of the last $\mathcal{X}$-Conv layer. The FC branch outputs are averaged during evaluation. The proposed model achieves 92.2% classification accuracy on ModelNet40 and 92.5% on rotation-aligned ModelNet40. An official implementation in TensorFlow is available [149].

Flex-Convolution [78] focuses on reducing computational and memory requirements, allowing large[9] point clouds to be processed. The simplified kernel is linear in each dimension – a dot-product of a trainable parameter vector and point's relative coordinates, with trainable bias added. The used point sub-sampling is random sampling weighted by local point cloud density. The authors achieve ModelNet40 classification accuracy of 90.2% (for 1024 points). Authors' implementation in TensorFlow is available [150].

Another continuous convolution with a spherical neighborhood kernel is presented by Liu et al. [79]. A 3-layer MLP computes the weights from a vector of low-level relations of the query point (centroid) and each point in the neighborhood: Euclidean distance, coordinate difference and point position. The classification network uses three convolutional layers and a MLP classifier. This model achieved ModelNet40 classification accuracy of 93.6% for 1024 points with voting across 10 random scales. An official implementation in PyTorch is available [151].

5.5 Sequential Point Cloud Processing

Processing the point cloud as a sequence with a suitable architecture such as a recurrent neural network or 1D convolutional network is also an option. However, an additional challenge needs to be overcome. As the order of points in a point cloud file can be arbitrary, the architecture must detect related points, or the input must be ordered so that spatially close points are near each other in the sequence.

5.5.1 Using Attention Mechanism

Attention mechanisms are widely used in natural language processing tasks such as machine translation [152]. The attention block produces a mask assigning a weight to each input element. In contrast to convolution, which exploits locality for feature extraction, attention can combine sequential elements regardless of their distance.

In **Point2Sequence** [80] the authors propose to use an attention-based encoder-decoder recurrent neural network to extract contextual information from local areas rather than hard-coding the neighborhood-merging operations. The proposed architecture first establishes multiscale areas by selecting centroids from the input point cloud by farthest point sampling and then samples points near the centroids at different scales. Then a per-point MLP and max pooling extract a 128-dimensional feature vector from each area. These features are then fed as a sequence to an LSTM [153] encoder. A decoder processes the encoder's representation while using an attention mechanism to aggregate the features from multiple scales and areas, producing a global 128-dimensional feature vector. Finally, a 3-layer, per-point MLP classifier produces the classification result. This architecture achieves ModelNet40 classification accuracy of 92.6%. An official TensorFlow implementation is available [154].

Set Transformer [81] is another encoder-decoder network utilizing attention for feature aggregation. Both the encoder and decoder consist of attention layers. To allow directly processing larger sets, the input set is first induced by few trainable parameters before computing attention, resulting in linear rather than quadratic complexity. After the input set is transformed by the *induced set attention* layers, the resulting feature vector is decoded by a multi-head-attention-based pooling layer with learnable seed points followed by a self-attention and a feed-forward layer. The authors report ModelNet40 classification accuracy of 90.4% with 5000 points. A PyTorch implementation is available [155].

5.5.2 Encoding Locality into the Order of Points

Another way of processing a set of points is to use the classical convolution operation but to change the order of input points to better reflect the object's spatial structure, so the spatially close points are close even in the representation.

Gadelha et al. [82] propose an encoder-decoder network to process a spatially-sorted list of points with strided 1D convolutions. The input points are assumed to be in the kd-tree order[10]. To enable learning both local and global features with convolution, the authors use a multi-resolution representation of the point cloud, passing three versions of the point cloud in each layer with a different number of samples – generated as a pre-processing step. To work with this representation, the authors propose a multi-resolution

9. Up to 7 million points can be processed concurrently.

10. Recursively constructed by sorting the sequence of points by position in a given axis, then splitting the sequence in half and recursively sorting the subsequences by the next axis, which is determined by randomly choosing an axis with a probability distribution based on the spans of points along axes.

convolution (MR-CONV) block, which takes a point cloud (with positions or general features) represented at three scales. It "mixes" the scales by average-pooling the higher-resolution point cloud, upsampling (nearest neighbour) the lower-resolution point cloud, and concatenating with the adjacent-resolution point clouds. Then the MR-CONV operation applies a strided 1D convolution, batch normalization and a non-linear activation function (ReLU). The classification network consists of an encoder (repeated MR-CONV operations) followed by an FC layer for classification. Apart from classification, the authors propose to use other encoder or decoder branches for other applications: image-to-shape inference and unsupervised shape reconstruction with latent representation learning. The proposed network achieves ModelNet40 classification accuracy of 91.7% in the supervised classification scenario[11] or 86.4% for classification based on features extracted using a variational autoencoder (VAE) trained on ShapeNet in unsupervised fashion. Official PyTorch implementation is available [156].

5.6 Unsupervised Learning of Shapes

As in 2D image-based and voxel-based neural networks, hourglass-shaped autoencoder-type networks can be trained in an unsupervised fashion to reconstruct the input, which forces the network to find a compressed *bottleneck* latent representation of the shape, from which the shape is reconstructed. This bottleneck representation (vector) can be used for classification by using it as a classifier input.

Achlioptas et al. [83] use an autoencoder consisting of PointNet-like encoder with five shared FC layers with ReLU activation and batch normalization, followed by max-pooling, and a decoder with 3 FC layers. This autoencoder is trained on the ShapeNet dataset with Chamfer distance used as a loss function to force the network to reconstruct a point cloud which both fits and covers the input. After the training, the encoder part is used for feature extraction, and a linear SVM classifier is used to classify the objects. This achieves ModelNet40 classification accuracy of 84.5%. An official TensorFlow implementation is available [157].

In **FoldingNet** [84], the authors also train an encoder-decoder network which reconstructs its input. However, instead of generating the output points implicitly, based solely on the bottleneck representation, the decoder is given a regular 2D grid to be folded to the desired shape, essentially fitting a surface to the given input shape. The encoder processes k-nearest-neighbor graph nodes and a per-point 3x3 covariance matrix of each point's neighbors' positions with MLPs (shared weights, applied per-point) and graph max-pooling layers, aggregating features from the neighborhood graph. The decoder transforms grid points coordinates using two MLP layers, each taking a concatenation of the bottleneck representation and a grid or intermediate point cloud (after the first folding). Chamfer distance is used as a loss function. A linear SVM classifier is used to classify the input point cloud based on the latent representation from the encoder, trained on the ShapeNet dataset in unsupervised fashion. The classification accuracy of 88.4% is achieved on the ModelNet40 dataset. Official PyCaffe implementation is available on request [158].

In addition to methods using the learned representation directly for a high-level task such as classification or segmentation, there is recent research exploring transfer learning in point cloud-based networks. Xie et al. [85] partially pre-train a network on a large dataset in an unsupervised fashion (finding correspondences between two differently transformed views of a point cloud) before fine-tuning it to the intended task. The authors report ShapeNetCore classification accuracy improvements between 0.6 pp and 5.9 pp reaching 85.7% when pre-training on ScanNet, compared to training from scratch. The greatest improvements are seen for least represented classes or when artificially limiting the dataset used for supervised training. While the code for segmentation experiments is available [159], the classifier code is not published.

6 Surface shape-based Neural Networks

An object's shape can be precisely or approximately represented by a set of freeform surfaces or polygonal faces with connectivity – a mesh. These representations are non-Euclidean, therefore operations such as convolution and pooling need to be re-defined in this domain. There are several ways of processing such data. The approaches in this category, commonly denoted *geometric deep learning*, work mainly with graphs and manifolds. In this section we briefly revise fundamental ideas of these approaches and survey approaches which assume mesh as their input format.

Manifold is a topological space where each point has a neighborhood that is topologically equivalent (homeomorphic) to an Euclidean space, called the tangent space [16]. Well-formed, watertight 3D surfaces have a manifold structure, locally homeomorphic to a 2D Euclidean space – a mesh is a 2D surface embedded in a 3D space. A metric and functions such as displacement can be defined on the manifold, enabling the use of convolution operation in the tangent space.

Graphs can be viewed as a special case of manifold – with a discrete domain and a neighborhood distance metric. Graph-based approaches were already introduced in Section 5.3, where graph convolution and subsampling were used to extract features from a point neighborhood graph constructed from a point cloud. Such a neighborhood graph assumes nearby points to be connected, which may not be true in general. E.g., when two unrelated faces from different parts of the object are near each other – this information is preserved in the mesh representation but is lost by conversion to a point cloud.

An extensive survey of fundamental graph- and manifold-based methods is presented in [16], which we refer the reader to for additional details about these approaches.

In addition to the general geometric approaches, there are approaches which attempt to define convolution and subsampling or pooling operations directly on meshes. These approaches are based on graph representation of the mesh, exploiting the "more regular" graph structure of triangular meshes, or define necessary operations for feature extraction and aggregation directly on a mesh.

11. 1K points sampled with Poisson disk sampling, voting was used during evaluation: the points are scale-augmented and their order is randomized 16 times by re-building the probabilistic kd-tree – the classification result is averaged.

6.1 Manifold-based Convolution

Manifold-based convolutional methods operate in the 2D manifold space of the object's surface. These methods mainly focus on correspondence or local feature extraction utilizing local charting, not classification, but such approaches can be applied to global feature extraction given a proper pooling operation.

Geodesic CNN [86] generalizes 2D convolution to Riemannian manifolds. The authors define a geodesic (charting) patch operator as a product of Gaussians using polar coordinates as parameters, producing a "blob" at a given distance and angle. Discretization of the operator to meshes is provided. The authors apply this approach to local descriptor extraction, shape correspondence and shape retrieval. Only the patch extraction Matlab code is available [160].

Anisotropic CNN [87] defines a different patch that is based on anisotropic heat diffusion which produces a directional "blob" with rotation *eccentricity* and scale given by the parameters. An official Theano implementation with preprocessing in Matlab is available [161].

Mixture Model CNN [88] generalizes classical Euclidean CNNs and previous approaches ([86, 87, 90]) as specific cases of a proposed unified framework. Both graph and manifold variants are covered. The authors also propose MoNet where the patch is a sum of learnable Gaussian kernels. The authors' implementation in Theano and Lasagne and TensorFlow with shape preprocessing in Matlab was unavailable at the time of writing[12] [162]. An alternative implementation of the graph-based variant is available as a part of the PyTorch Geometric framework [21].

6.2 Graph-based Convolution

Graph-based approaches are based on propagating signals on a graph by filtering in spatial or spectral domain. There are two main related tasks commonly tackled in graph-structured data such as social networks: node classification (classification of each graph node, corresponding to segmentation or local feature extraction), and graph classification (global feature extraction), producing single output for the whole graph.

Bruna et al. [89] define graph convolution and pooling operations generalization in spatial or spectral construction. In the spatial construction, each convolutional layer computes a dot product of node features with learnable filters and then pools values over nodes in a cluster, which can be done repeatedly in a hierarchical manner. No weight sharing within a layer is used. The spectral construction operates on eigenvectors of the Laplacian matrix [163] of the graph. Only a limited number of eigenvectors are processed; this corresponds to low-pass filtering. An unofficial implementation of the convolution operator is available [164].

Diffusion-convolutional Neural Networks [90] extract the context of each node by graph diffusion – spreading node features by matrix power series up to a pre-defined hop count. No pooling operation is provided. An official implementation in Theano and Lasagne is available [165].

Graph Convolutional Network by Kipf and Welling [91] shows a propagation rule which approximates spectral graph convolution, diffusing local features to neighboring nodes. As with Euclidean convolution, the receptive field (or propagation hop count) increases with the number of layers. The layer weights are shared across nodes. The network structure remains unchanged as no pooling operation is used. The authors provide their TensorFlow implementation [166].

Graph-CNN by Dominguez et al. [92] uses a graph-convolutional approach for classifying a mesh converted to a graph. After mesh face count reduction, the authors create 8 adjacency matrices, each containing edges with a specific axis-aligned orientation (e.g. positive X and Y direction and negative Z). This data is processed by several graph convolutional layers, each of which increases the receptive field of features stored in nodes. No pooling layer is used; the authors believe that "small increases in receptive field size through cascaded convolution layers will be enough for effective inference" [92, Sec. 3.1]. This approach's classification accuracy is evaluated on a mesh dataset ModelNet10[13] achieving 74.3% with 4 layers each with 24 filters. No implementation was found at the time of writing.

FeaStNet [93] presents a graph-convolution operator generalizing classical convolution. Rather than using a fixed weight for all neighbors, the weight of filters for a neighboring node is determined by a *soft-assignment* function using a linear transformation of node features with learned weights, passed through soft-max over all neighbors which decouples the learned weights from the neighbor count. The authors focus on 3D shape correspondence and part segmentation, but with suitable pooling, the approach could be applied to classification. A TensorFlow implementation is available [167].

6.3 Native Mesh-based Approaches

Approaches in this section either use the mesh directly as an internal representation format, defining operations directly on the mesh, or assume mesh structure constraints for processing the mesh using a generic graph approach.

Directionally convolutional networks for 3D shape segmentation [94] process the mesh similarly to a graph. The input is a triangular mesh with face normal vectors used as features. Convolution and pooling operations are defined in the spatial domain using *rings* – a set of neighboring faces at a given distance. Faces in each ring are ordered counterclockwise, starting from the maximum curvature direction. The proposed *directional convolution* operator is a normalized dot product of ordered neighboring faces' features and a learned kernel. Pooling operation downsamples a given number of rings (a cluster) into one face, computing the maximum or the average of each feature over the cluster. The proposed model outputs segmentation – a class for each face – with a network combining a global fully connected branch and a local branch containing two *directional convolution* and pooling layers, and a MLP. No implementation was found at the time of writing.

12. An archived version cached by web.archive.org was available.

13. A smaller subset of the ModelNet dataset – the accuracy is not directly comparable with results on ModelNet40, but usually MN40 accuracies are lower than MN10 for the same classification method.

Surface Networks [95] provide a graph neural networks improvement suited for triangular meshes. The authors propose using the Dirac operator rather than the Laplacian to extract the differential information from features stored in vertices or faces. Dirac operator is a first-order differential operator capable of extracting principle curvature directions; Laplacian is a second-order operator with access only to local mean curvature and normal vector. The proposed networks consist of alternating layers where the Dirac operator or its adjoint is applied to either vertices or faces. The authors use the proposed layers in a deep VAE architecture, using 15 modified ResNet-v2 blocks with convolution replaced by Laplace or Dirac operators. They focus on shape reconstruction; classification performance has not been evaluated. Reference PyTorch implementation is available [168].

MeshNet [96] uses a similar approach as point cloud-based networks: processing a set of elements constructed from faces by per-face feature extraction. Relative triangle corner positions and a normal vector direction, and information about neighboring faces are used to extract per-element features. The spatial and structural features are extracted and processed separately. In the proposed classification network, elements are fed through two proposed *Mesh Conv* layers, which transform concatenated features from each face and its neighboring faces by a MLP, and "inflated" by a shared MLP. Then a global feature vector is computed by another MLP processing concatenated outputs of both *Mesh Conv* layers, and pooling over all faces. Finally, a MLP classifier produces class scores. This approach achieves ModelNet40 classification accuracy of 91.9%, comparable to point cloud-based networks. An implementation in PyTorch is provided [169].

MeshCNN [97] proposes to store features in edges of a triangular mesh. Initially, the following edge features are extracted: dihedral angle, two inner angles and two distances to the opposite vertex relative to the edge length. Pooling is implemented as edge collapse into a point, merging the then-overlapping pairs of edges. This approach requires the input to be a manifold mesh. Convolution is defined as a sum of features of an edge and its four adjacent edges' symmetrized features weighted by a learned kernel. The approach is used in classification and segmentation architectures. In the case of classification, averaging the feature vectors of remaining edges is then used as a symmetric function to produce a global feature vector, which is then fed to a dense classifier. The proposed classification architecture has been evaluated on a smaller SHREC dataset [170], but not on the ModelNet dataset as it contains non-manifold shapes. The authors provide a PyTorch implementation [171].

7 Hybrid Approaches

Ensembling or merging different models has been proved to improve classification accuracy [172]. In case of 3D shape classification, different classification models tend to have different failure cases depending on which aspects of the shape are captured by the used representation. Volumetric representation captures the high-level shape or spatial distribution of the object but lacks details due to limited resolution. Multi-view image-based representation better captures silhouettes and surface curvature captured by shading, but is limited to two dimensions, which causes loss of details due to occlusion. Point cloud representation may provide additional information about surface shape in the object's parts which are not visible from the selected viewpoints while lacking the curvature information. Mesh-based representation retains connectivity information, which is not present in the point cloud. Combining classification methods that use different representations may improve the resulting classifier's robustness, similarly to the multi-view image-based networks that use different modalities (surveyed in Section 4.2). In this section we focus on approaches using a combination of different 3D object representations.

In autonomous driving research, the fusion of an RGB camera (2D) and LiDAR (3D) data is commonly done (e.g. [173]). Compared to the autonomous driving context where a single viewpoint is used (or a few viewpoints from the vehicle), the 3D shape classification task is not limited to the images captured from the vehicle position but instead can use arbitrary views of the object.

We identified two main approaches to combining networks that use different input representations: *ensembling* – combining the classification scores e.g., by computing an element-wise average – which allows combining any type of classification methods to produce one, more robust prediction, and *descriptor merging*, which merges intermediate feature vectors and then trains a common classifier.

7.1 Ensembling

FusionNet [98] is an ensemble of a 20-view MVCNN processing grayscale images and two 2D convolutional volumetric networks processing voxel grids of 60 rotations of the object about the gravity axis. Both volumetric networks take $30 \times 30 \times 30$ occupancy grid as an input, and treat one of the spatial dimensions as feature channels (like color) of the remaining two dimensions, on which 2D convolutions are applied. The first convolutional network (V-CNN I) has a common 2D CNN classification architecture with three convolutional and two fully connected layers. In V-CNN II, the first two convolutions are replaced by Inception-like modules. The multi-view CNN is based on AlexNet, pre-trained on ImageNet. Each network has only one set of weights shared among all the views; features from individual views are max-pooled, and all the networks are trained separately. This ensemble achieves 90.80% ModelNet40 accuracy. No code was found at the time of writing.

Su et al. [13] experiment with ModelNet40 subsets, reducing the number of objects per category. In this experiment, the authors use various ensembles of MVCNN, VoxNet and PointNet. An ensemble averaging classification scores of VoxNet and PointNet achieves a better classification accuracy around 90% on the whole ModelNet40, which is higher than the individual networks (between 85% and 90%). An ensemble of all three networks did not provide any significant accuracy increase compared to just using MVCNN. The authors also train a linear model on concatenated features extracted by the networks, which they report did not provide any improvements.

Koguciuk et al. [20] evaluate feasibility of ensembling seven different 3D shape classification methods using point

cloud as an input. While all ensembled methods use the same representation, the authors measure classification accuracy improvements using different architectures compared to using just one architecture trained on different data as in [27]. A ModelNet40 classification accuracy of 94.03% is reported using an ensemble of two architectures: SO-Net and PointCNN, and 94.15% using Pointnet, PointNet++ and SO-Net. The code is available [174].

7.2 Descriptor Merging

PVNet [99] combines point cloud and multi-view representation, extracting features from both using suitable networks. The point cloud is transformed by a learned matrix similar to PointNet's input transform *T-net*. The features are then extracted by a sequence of two *edge convolutions* – k-nearest neighbours, 1x1 convolution and pooling. The features from multi-view images are produced by a CNN with weights shared among all the views. The pooled view features are transformed to the same subspace as point cloud features by an embedding layer. An *embedding attention fusion* scheme merges the features from both branches. The per-point features pass through two *attention fusion* blocks with access to the embedded view-based feature vector, choosing significant local features by a learned attention mask from global view features. The point cloud features pass through the convolution and pooling layer and are concatenated with the embedded view-based features. Finally, this vector is classified by a FC classifier. This method achieves 93.2% ModelNet40 accuracy using 1024 points and 12 views with AlexNet used as the multi-view network feature extractor, pre-trained on ImageNet. The individual multi-view network alone achieves 89.9% accuracy, the point-cloud part reaches 92.2%. When combined naively by concatenation before the last FC layer, only 92.5% accuracy is achieved, showing benefit of the merging scheme. An implementation in PyTorch is available [175].

8 Evaluation Methodology

In this section, we describe how we train and evaluate performance of several neural network classification models on the same datasets and compare their results. We use two datasets in our performance evaluation; for all but two networks, we extend the evaluation by measuring how the methods behave on more general data than originally used by the authors.

8.1 Datasets

Since we aim to compare the classification performance of multiple network types on the same set of 3D shapes, our choice is limited – rendering (conversion to images for the multi-view image-based methods) requires at least approximate surfaces as input. From the formats of available datasets, mesh is the most suitable to be converted to the other representations as it provides the most information about the object surface. Using point-cloud-based datasets would require surface reconstruction introducing artifacts; similarly, image-based datasets lack spatial information preventing conversion to a point cloud or a volumetric representation. Therefore the input to our evaluation framework is a set of labeled 3D meshes, which we convert into each network's native input format using methods described below. In addition to vertices and faces, 3D mesh files can contain information such as material description and texture coordinates, but we ignore these as the chosen classification methods do not utilize them.

We use two mesh datasets with sufficient size and quality: ModelNet40 and ShapeNetCore.

ModelNet [26] is one of the most known and commonly used datasets containing annotated 3D models in a mesh format. Its subset, ModelNet40 with 12,311 models in 40 categories covering mainly furniture, household objects and appliances, is intended as a benchmark for testing different machine learning approaches. We use this dataset as the main focus for our evaluations. The dataset has an official $80\% : 20\%$ split to training and testing subsets. We split a validation subset from the training subset in experiments prefixed *modelnet40val-*. ModelNet40 models have widely different scales, therefore we normalize each model's scale to fit inside a unit sphere (for point clouds) or unit cube (volumetric grids and images). For additional experiments detailed in the supplementary material we also use a manually aligned version of the dataset [29].

ShapeNet [176] is an ongoing effort to establish a richly-annotated, large-scale dataset of 3D shapes. We use its subset called ShapeNetCore (v2), providing 51,209 models in 55 categories, partially overlapping with ModelNet40 categories, but including more appliances, vehicle types and other objects. Less than 2.5% of the models are assigned to more than one category or are duplicates, so we excluded those[14], yielding 50,769 unique models, each in exactly one category. While there is an official split to training, test, and validation sets for the dataset version used in the SHREC16 competition, it does not contain all current models and is not divided uniformly within categories. We, therefore, construct our own split randomly choosing 70% of models in each category assigned to the training set, 10% to validation and the rest to the test set. Our final split into sets and categories is available on this paper's webpage; an overview of the split within each category is shown in the supplementary material.

Using just the train/test split and choosing the hyperparameters such as training epoch count based on the test set accuracy is biased. This issue is commonly solved by cross-validation, which requires running the training and evaluation multiple times, or by introduction of a third subset (split into train/validation/test), which would be used for final evaluation once the optimal hyperparameters are found and fixed on the train and validation subsets, not containing the *held out* test samples. We employ the latter option for selecting the training epoch count in this article. In contrast, using the train/test split allows the use of the most data for training and evaluation, minimizing the error in computing the accuracy – we include additional experiments in the supplementary material which use the full train/test split.

14. Exception: all models in the *cellular_telephone* category are also in the *telephone* category. We kept such models in the former category.

8.2 Data Conversion

Classification methods use different input representations, therefore we have to convert meshes to voxels, images, and point clouds. Examples of different conversion method outputs are presented in the supplementary material.

Voxel occupancy grids are created from ModelNet40 and ShapeNetCore meshes by voxelization using the OpenVDB library [177]. Each object may be voxelized multiple times with the source mesh rotated along the vertical axis as an augmentation method proposed by [25]. Voxelization for the **O-CNN** and **Adaptive O-CNN** networks is done using the authors-provided tools.

Images for the multi-view image-based neural networks are rendered in twelve rotation steps, following [44]. In our evaluation, the same set of images is used for each classification method to ensure a fair comparison. We render two kinds of images: *shaded* and *depth* images produced by the code by Su et al. [13].

Point clouds are constructed from 3D meshes by sampling points on the mesh surface. We experiment with different sampling methods: *uniform*, *Lloyd* and *Poisson*. A point in *uniform* sampling is generated by first selecting a polygon with a probability proportional to its area and choosing a point within the polygon by generating random barycentric coordinates. To reduce point clustering, we employ *Lloyd*'s algorithm [178] from Point Cloud Utils library [179], which samples the mesh more regularly. We also use *Poisson*-sampled point clouds similar in distribution to the data provided by Qi et al. [132], which were generated by farthest point sampling of a dense, uniformly-sampled point cloud, and which we use as reference. Following [132], all point clouds contain 2048 points. Training and evaluation use all points, except in the replication experiments marked *-qi1024* using only 1024 points.

Visualizations of outputs of the used conversion methods and the discussion on their impact on the achieved accuracy are in the supplementary material.

8.3 Testing Scope

We limit the scope of our testing in the following ways. We only consider single-object classification, although several of the networks can be extended to perform part segmentation as well as new shape generation. We do not test any large ensembles of networks [27, 28, 58] as our focus is to compare the individual methods' performance. We also consider only networks with publicly available code as implementing all the different techniques is beyond the scope of this work.

To compare with previously reported results, we compute overall (per-instance) classification accuracy as a main performance metric. We report the accuracy on test set at the epoch number where the validation accuracy was the highest. We also include the highest accuracy achieved on the test set and the validation set as separate data points.

8.4 Technical Setup

We conduct all our experiments on Linux machines with AMD RYZEN 1950X or two Intel Xeon E5-2680 v3 CPUs, 128 GB of RAM and NVIDIA GeForce GTX 1080 Ti GPUs with driver version 440.44. We only use one GPU per network to compare the training time of different models since only some support multi-GPU training. To reduce software dependencies of our framework, we use Docker [180] to enclose the networks and their dependencies into self-contained containers. Thus, every neural network and data conversion tool is an independent piece of software, which can be run on any system with Docker and NVIDIA Container Toolkit[15] installed. For information about the prerequisites and structure of our framework, please consult the manual of our code.

9 Results

In this section we present the replication results of selected classification approaches (see Table 1) as we run training experiments on ModelNet40. Besides, we evaluate how rotational alignment impacts the accuracy of the methods. We extend the training to a larger dataset (ShapeNetCore) to see how the approaches generalize to a dataset with more categories and more training examples. The datasets are described in Section 8.1.

For the evaluation, we selected networks which are significant by either being the first to process a given shape representation or those which achieved high reported accuracy and the author's implementation was available.

Voxel-based approaches are represented by a 3D convolutional network with residual connections **Voxception-ResNet** [27][16] (*vrn*), and octree-based approaches **O-CNN** and **Adaptive O-CNN** [34, 110] (*octree* and *octree_adaptive*). From multi-view approaches, we use *mvcnn2* [13], *rotnet* [51], *seq2seq* [50], and *vgg*, our MVCNN-like network implementation based on VGG19 extractor-classifier [181] with voting among the views. Selected point cloud-based approaches are **PointNet** by Qi et al. [61] (*pointnet*), **PointNet++** [63] (*pointnet2*), **Kd-networks** [65] (*kdnet*) and **SO-Net** [64] (*sonet*).

The results are shown in Figure 6. A larger figure, a table with exact values, and additional training details are available in the supplementary material.

9.1 Replication

When training, we did not reach the reported accuracies in some cases. The accuracy decrease can be partially attributed to using a less data for training as a part of the dataset is used as validation subset. We preferred to use the same data conversion method for as many networks as possible within a shape representation type, for a meaningful comparison of networks. To measure the impact of different data conversion, we also train on the original datasets for comparison, if provided by the authors.

Different dataset conversion methods impact the classification accuracy. This is most clearly visible for point cloud-based networks where using *Poisson*-sampled point clouds yielded ModelNet40 and ShapeNetCore test set accuracies 0.7 pp and 0.51 pp, respectively, higher than *uniform* sampling. When using the ModelNet40 point clouds

15. https://github.com/NVIDIA/nvidia-docker

16. Due to high computation demands of training we chose a single architecture: Voxception-ResNet with the highest reported accuracy.

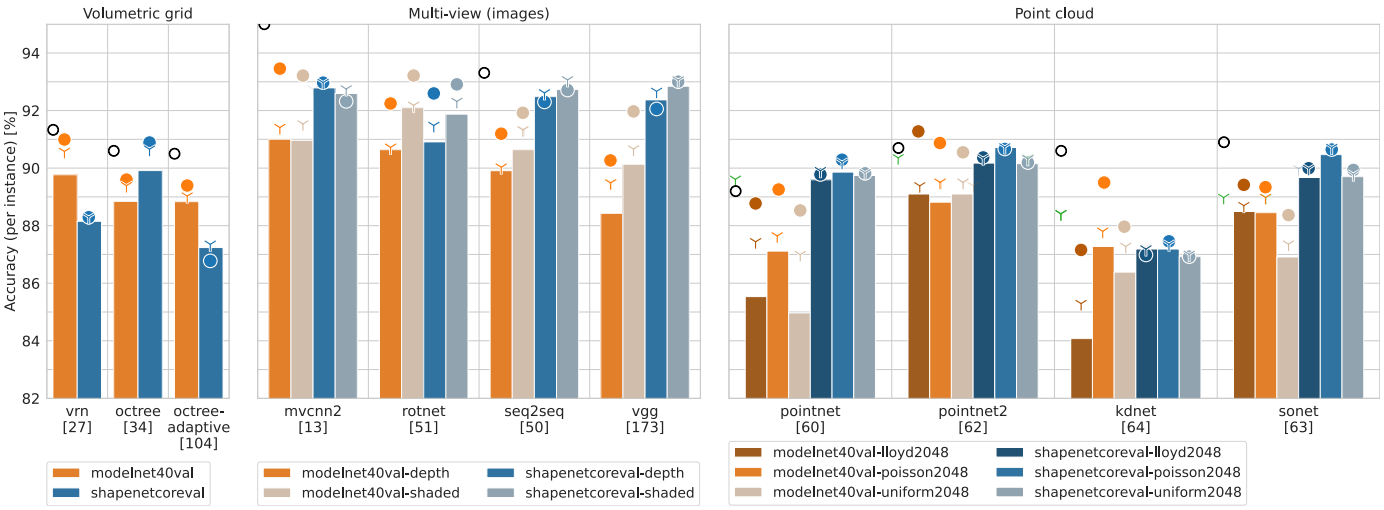


Fig. 6. The measured accuracies on different datasets. The bars show the test set accuracy at the epoch with the best validation accuracy and same-colored ϒ and ● mark the highest achieved accuracy on the test and validation subsets on the same dataset, respectively. ○ marks accuracies reported on ModelNet40 and ϒ marks the replicated test set accuracy on the point clouds provided by Qi et al. [61] using 1024 points (*qi1024*).

provided by Qi et al. [61][17], the achieved accuracies on the provided test set are comparable to the reported accuracies and also comparable to our accuracies on the validation set. The maximum test accuracy being always lower for ModelNet40 (but not ShapeNetCore) than the maximum validation accuracy could indicate a different distribution of the 3D shapes present in the official train and test subsets of ModelNet40. We have not verified whether the data from [61] use the official train/test split. Image-based data conversion may differ in the used renderer or rasterizer, camera or lighting settings, material or shading. However, despite using the authors' provided rendering tools [13], we reach a maximum ModelNet40 test set accuracy of only 91.53% for *mvcnn2* instead of the reported maximum of 95.0%. By replicating volumetric grid-based networks, we reach on average 1.14 pp lower maxima than reported. In the case of *vrn* we use a different voxelization library; other networks use authors' provided conversion tools.

Apart from input data size, quality, and dataset split, the results may be influenced by non-determinism of the training (weights optimization) process. This includes random initialization of the network weights, possibly different random order of training examples and variations in hyperparameters such as batch size, leading to different mini-batches, as well as non-determinism caused by hardware parallelism. All these factors may lead the optimization process to find a different local minimum or plateau of the loss function. The impact of these non-determinism causes can be trivially reduced by running each experiment multiple times; unfortunately, we did not have the necessary computational capacity.

Note that there were also some hyperparameter and input differences to the training made due to unclear training or evaluation procedure or data compatibility. For *vrn*, the exact training process was not described. The authors report their best-performing models were first trained on 12 voxelization views and then fine-tuned on 24 views. We trained a 12-view to match the authors-provided data. For the point-cloud networks which accept normal vectors (*pointnet2* and *sonet*), we do not provide them to the network so we can compare performance on exactly the same point clouds for all point-cloud methods. In *kdnet*, we use the shallower network variant with 10 layers.

17. Different sampling of points and post-processing: sampling 10k points, then farthest point sampling to select fewer points with better coverage of the surface.

9.2 Overall Results

On average, using all datasets in our experiments, multi-view image-based approaches reached the highest classification accuracy (91.40% ± 1.27 pp and 91.76% ± 1.11 pp at the best test and train epoch, respectively), followed by volumetric grid-based approaches (88.79% ± 1.00 pp and 89.21% ± 1.28 pp) and point cloud-based approaches (88.23% ± 1.84 pp and 88.71% ± 1.50 pp).

While *vgg* reached the maximum (test set) accuracy of 92.84% and 93.06% at the best validation and test epochs, respectively, with averages 90.94% ± 2.05 pp and 91.46% ± 1.68 pp, *mvcnn2* yielded the highest average accuracies of 91.83% ± 0.98 pp and 92.16% ± 0.81 pp among multi-view image-based networks. The *rotnet*'s respectable average accuracy of 91.38% ± 0.71 pp and 91.67% ± 0.73 pp is achieved with approximately half the computational and memory cost. The classification accuracy of *seq2seq* is similar to *vgg* from which it uses extracted features: on average better (by 0.50 pp and 0.29 pp) but with lower maximum accuracy in some cases. This shows the benefit of view fusion using RNN being more robust than a simple max pooling.

In the volumetric grid-based networks, *octree* achieved the best accuracy: both maximum (89.91% and 90.65%) and on average (89.37% ± 0.75 pp and 90.02% ± 0.89 pp). The *vrn* and *octree-adaptive* networks reached slightly lower average accuracies (88.95% ± 1.14 pp and 89.43% ± 1.61 pp for *vrn* and 88.03% ± 1.12 pp and 88.18% ± 1.18 pp for *octree-adaptive*, respectively), while being two orders of magnitude faster to train – see the time and memory measurements in the supplementary material.

Among point cloud-based methods, *pointnet2* yields the highest maximum accuracy of (90.71% and 90.73% on the best validation and test epoch, respectively) and the best average with the lowest standard deviation (89.67% $\pm$ 0.76 pp and 89.67% $\pm$ 0.56 pp). It is closely followed by *sonet* (88.95% $\pm$ 1.26 pp and 89.25% $\pm$ 1.17 pp with maxima of 90.46% and 90.71%) and *pointnet* (87.80% $\pm$ 2.23 pp and 88.67% $\pm$ 1.45 pp with maxima of 89.86% and 90.20%) The *kdnet* network, which appears to behave better with larger point clouds in our experiments, has in this case lower average accuracy of 86.50% $\pm$ 1.23 pp and 86.98% $\pm$ 0.67 pp.

9.3 The Impact of Dataset Size

Comparing the training performance on ShapeNetCore to the ModelNet40 dataset, most networks' accuracy improves (on average by 1.64 pp and 1.24 pp for test accuracies at the best validation and test epoch, respectively) with two volumetric grid-based methods as exceptions: *octree-adaptive* and *vrn* networks achieve a lower accuracy on the larger dataset with more object categories, possibly due to limited capacity or because their hyperparameters may have been optimized by the authors for the smaller ModelNet40 dataset. Image-based multi-view networks show the most consistent increase in accuracy (+1.85 pp and +1.7 pp). These networks have the capacity to internally learn more features as the image-based models are largest of all the representations.

10 Conclusions

We surveyed 76 neural networks which process 3D shapes, created a taxonomy to categorize them based on the type of shape representation they process, and grouped individual approaches by common ideas. We observe a rise mainly in the point cloud-based method research. Xie et al. [85] show possible performance improvements using transfer learning for point cloud methods. This may bring significant performance improvements seen in image-based transfer learning also to 3D shape representation learning. We expect to see more research in this direction.

For evaluation, we selected 11 classification approaches and compared them on two datasets converted by different conversion methods. We saw that the data conversion method (e.g., point cloud sampling or image rendering method) can have a large impact on the achievable classification accuracy. We observe multi-view image-based representations yielding the best classification accuracy.

The code we used to convert the datasets, train the networks, and evaluate their performance is packaged to minimize the software prerequisites and is available on our project's website.[18] We believe this work provides a useful overview of existing methods and will simplify running machine learning experiments on 3D shapes.

Bibliography

[1] A. Uçar, Y. Demir, and C. Güzeliş, "Object recognition and detection with deep learning for autonomous driving applications," *Simulation*, vol. 93, no. 9, 2017.

[2] A. Meyer-Baese and V. J. Schmid, *Pattern recognition and signal analysis in medical imaging*. Elsevier, 2014.

[3] C.-h. Chen, *Handbook of pattern recognition and computer vision*. World Scientific, 2015.

[4] N. Wang, Y. Zhang, Z. Li, Y. Fu, W. Liu, and Y.-G. Jiang, "Pixel2Mesh: Generating 3D mesh models from single RGB images," in *ECCV*, September 2018.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *NIPS*. Curran Associates Inc., 2012.

[6] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations, ICLR*, 2015.

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *arXiv:1512.03385*, Dec. 2015.

[8] A. Zelener, "Survey of object classification in 3D range scans," *Technical report*, 2015.

[9] A. Ioannidou, E. Chatzilari, S. Nikolopoulos, and I. Kompatsiaris, "Deep learning advances in computer vision with 3D data: A survey," *ACM Computing Surveys (CSUR)*, vol. 50, no. 2, 2017.

[10] L. Carvalho and A. von Wangenheim, "3D object recognition and classification: a systematic literature review," *Pattern Analysis and Applications*, vol. 22, no. 4, 2019.

[11] E. Arnold, O. Y. Al-Jarrah, M. Dianati, S. Fallah, D. Oxtoby, and A. Mouzakitis, "A survey on 3D object detection methods for autonomous driving applications," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 10, 2019.

[12] D. Griffiths and J. Boehm, "A review on deep learning techniques for 3D sensed data classification," *Remote Sensing*, vol. 11, no. 12, 2019.

[13] J.-C. Su, M. Gadelha, R. Wang, and S. Maji, "A deeper look at 3D shape classifiers," *arXiv:1809.02560*, Sep. 2018.

[14] X. Shen, "A survey of object classification and detection based on 2D/3D data," *arXiv:1905.12683*, 2019.

[15] C. Wang, M. Cheng, F. Sohel, M. Bennamoun, and J. Li, "NormalNet: A voxel-based CNN for 3D object classification and retrieval," *Neurocomputing*, vol. 323, 2019.

[16] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: going beyond euclidean data," *IEEE Signal Processing Magazine*, vol. 34, no. 4, 2017.

[17] R. Rostami, F. S. Bashiri, B. Rostami, and Z. Yu, "A survey on data-driven 3D shape descriptors," in *Computer Graphics Forum*, vol. 38, no. 1. Wiley Online Library, 2019.

[18] E. Ahmed, A. Saint, A. E. R. Shabayek, K. Cherenkova, R. Das, G. Gusev, D. Aouada, and B. Ottersten, "A survey on deep learning advances on different 3D data representations," *arXiv:1808.01462v2*, 2019.

[19] R. D. Singh, A. Mittal, and R. K. Bhatia, "3D convolutional neural network for object recognition: a review," *Multimedia Tools and Applications*, vol. 78, no. 12, 2019.

[20] D. Koguciuk, Ł. Chechliński, and T. El-Gaaly, "3D

18. https://cgg.mff.cuni.cz/~martinm/papers/2021-survey-eval

object recognition with ensemble learning—a study of point cloud-based deep learning models," in *International Symposium on Visual Computing*. Springer, 2019.

[21] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019. [Online]. Available: https://github.com/rusty1s/pytorch_geometric

[22] M. Yavartanoo, E. Y. Kim, and K. M. Lee, "SPNet: Deep 3D object classification and retrieval using stereographic projection," in *Asian Conference on Computer Vision*. Springer, 2018.

[23] N. Vukašinović and J. Duhovnik, *Introduction to Freeform Surface Modelling*. Springer International Publishing, 2019.

[24] Q. Ji and M. M. Marefat, "Machine interpretation of CAD data for manufacturing applications," *ACM Comput. Surv.*, vol. 29, no. 3, 1997.

[25] D. Maturana and S. Scherer, "VoxNet: A 3D convolutional neural network for real-time object recognition," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015.

[26] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3D ShapeNets: A deep representation for volumetric shapes," *arXiv:1406.5670*, Jun. 2014.

[27] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "Generative and discriminative voxel modeling with convolutional neural networks," *arXiv:1608.04236*, Aug. 2016.

[28] V. Arvind, A. Costa, M. Badgeley, S. Cho, and E. Oermann, "Wide and deep volumetric residual networks for volumetric image classification," *arXiv:1710.01217*, Sep. 2017.

[29] N. Sedaghat, M. Zolfaghari, E. Amiri, and T. Brox, "Orientation-boosted voxel nets for 3D object recognition," *arXiv:1604.03351*, Apr. 2016.

[30] S. Zhi, Y. Liu, X. Li, and Y. Guo, "LightNet: A lightweight 3D convolutional neural network for real-time 3D object recognition," in *Eurographics Workshop on 3D Object Retrieval*. The Eurographics Association, 2017.

[31] X. Xu and S. Todorovic, "Beam search for learning a deep convolutional neural network of 3D shapes," in *CVPR*, Dec. 2016.

[32] C. Ma, Y. Guo, Y. Lei, and W. An, "Binary volumetric convolutional neural networks for 3-D object recognition," *IEEE Transactions on Instrumentation and Measurement*, vol. 68, no. 1, Jan. 2019.

[33] G. Riegler, A. O. Ulusoy, and A. Geiger, "OctNet: Learning deep 3D representations at high resolutions," in *CVPR*, Jul. 2017.

[34] P.-S. Wang, Y. Liu, Y.-X. Guo, C.-Y. Sun, and X. Tong, "O-CNN: Octree-based convolutional neural networks for 3D shape analysis," *ACM Transactions on Graphics (SIGGRAPH)*, vol. 36, no. 4, 2017.

[35] ——, "Adaptive O-CNN: A patch-based deep representation of 3D shapes," *ACM Transactions on Graphics (SIGGRAPH Asia)*, vol. 37, no. 6, 2018.

[36] A. Sharma, O. Grau, and M. Fritz, "VConv-DAE: Deep volumetric shape learning without object labels," in *ECCV*. Springer, 2016.

[37] J. Wu, C. Zhang, T. Xue, B. Freeman, and J. Tenenbaum, "Learning a probabilistic latent space of object shapes via 3D generative-adversarial modeling," in *NIPS*. Curran Associates, Inc., 2016.

[38] W. Wang, Q. Huang, S. You, C. Yang, and U. Neumann, "Shape inpainting using 3D generative adversarial network and recurrent convolutional networks," in *2017 IEEE International Conference on Computer Vision (ICCV)*, Oct. 2017.

[39] S. Liu, L. Giles, and A. Ororbia, "Learning a hierarchical latent-variable model of 3D shapes," in *2018 International Conference on 3D Vision (3DV)*, Sep. 2018.

[40] Y. Li, S. Pirk, H. Su, C. R. Qi, and L. J. Guibas, "FPNN: Field probing neural networks for 3D data," in *NIPS*, 2016.

[41] A. Garcia-Garcia, F. Gomez-Donoso, J. Garcia-Rodriguez, S. Orts-Escolano, M. Cazorla, and J. Azorin-Lopez, "PointNet: A 3D convolutional neural network for real-time object class recognition," in *2016 International Joint Conference on Neural Networks (IJCNN)*, Jul. 2016.

[42] T. Le and Y. Duan, "PointGrid: A deep network for 3D shape understanding," in *CVPR*, Jun. 2018.

[43] Y. Lecun and F. J. Huang, "Learning methods for generic object recognition with invariance to pose and lighting," in *Proceedings of CVPR'04*. IEEE Press, 2004.

[44] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, "Multi-view convolutional neural networks for 3D shape recognition," *arXiv:1505.00880*, May 2015.

[45] E. Johns, S. Leutenegger, and A. J. Davison, "Pairwise decomposition of image sequences for active multi-view recognition," in *CVPR*, Jun. 2016.

[46] L. Minto, P. Zanuttigh, and G. Pagnutti, "Deep learning for 3D shape classification based on volumetric density and surface approximation clues," in *Proceedings of the 13th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*. SCITEPRESS - Science and Technology Publications, 2018.

[47] P. Zanuttigh and L. Minto, "Deep learning for 3D shape classification from multiple depth maps," in *2017 IEEE International Conference on Image Processing (ICIP)*, Sep. 2017.

[48] K. Sarkar, B. Hampiholi, K. Varanasi, and D. Stricker, "Learning 3D shapes as multi-layered height-maps using 2D convolutional networks," in *The European Conference on Computer Vision (ECCV)*, Sep. 2018.

[49] Y. Feng, Z. Zhang, X. Zhao, R. Ji, and Y. Gao, "GVCNN: Group-view convolutional neural networks for 3D shape recognition," in *CVPR*, Jun. 2018.

[50] H. Zhizhong, S. Mingyang, and L. Zhenbao, "SeqViews2SeqLabels: Learning 3D global features via aggregating sequential views by RNN with attention," *IEEE Transactions on Image Processing*, vol. 28, no. 2, Sep. 2018.

[51] A. Kanezaki, Y. Matsushita, and Y. Nishida, "RotationNet: Joint object categorization and pose estimation using multiviews from unsupervised viewpoints," *arXiv:1603.06208*, Mar. 2018.

[52] K. Sarkar, E. Mathews, and D. Stricker, "Structured 2D representation of 3D data for shape processing,"

arXiv:1903.10360, Mar. 2019.

[53] A. A. Soltani, H. Huang, J. Wu, T. D. Kulkarni, and J. B. Tenenbaum, "Synthesizing 3D shapes via modeling multi-view depth maps and silhouettes with deep generative networks," in *CVPR*, Jul. 2017.

[54] Z. Han, X. Liu, Y.-S. Liu, and M. Zwicker, "Parts4Feature: Learning 3D global features from generally semantic parts in multiple views," *arXiv:1905.07506*, May 2019.

[55] A. Sinha, J. Bai, and K. Ramani, "Deep learning 3D shape surfaces using geometry images," in *Computer Vision – ECCV*. Springer International Publishing, 2016, vol. 9910.

[56] B. Shi, S. Bai, Z. Zhou, and X. Bai, "DeepPano: Deep panoramic representation for 3-D shape recognition," *IEEE Signal Processing Letters*, vol. 22, no. 12, Dec. 2015.

[57] K. Sfikas, T. Theoharis, and I. Pratikakis, "Exploiting the PANORAMA Representation for Convolutional Neural Network Classification and Retrieval," in *Eurographics Workshop on 3D Object Retrieval*. The Eurographics Association, 2017.

[58] K. Sfikas, I. Pratikakis, and T. Theoharis, "Ensemble of PANORAMA-based convolutional neural networks for 3D model classification and retrieval," *Computers & Graphics*, vol. 71, Apr. 2018.

[59] Z. Cao, Q. Huang, and R. Karthik, "3d object classification via spherical projections," in *2017 International Conference on 3D Vision (3DV)*, Oct. 2017.

[60] C. Esteves, C. Allen-Blanchette, A. Makadia, and K. Daniilidis, "Learning SO(3) equivariant representations with spherical CNNs," in *Computer Vision – ECCV*. Springer International Publishing, 2018, vol. 11217.

[61] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3D classification and segmentation," in *CVPR*, 2017.

[62] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola, "Deep Sets," in *NIPS*. Curran Associates, Inc., 2017.

[63] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "PointNet++: Deep hierarchical feature learning on point sets in a metric space," *arXiv:1706.02413*, Jun. 2017.

[64] J. Li, B. M. Chen, and G. H. Lee, "SO-Net: Self-organizing network for point cloud analysis," *arXiv:1803.04249*, Mar. 2018.

[65] R. Klokov and V. Lempitsky, "Escape from cells: Deep Kd-Networks for the recognition of 3D point cloud models," *arXiv:1704.01222*, Apr. 2017.

[66] M. Dominguez, R. Dhamdhere, A. Petkar, S. Jain, S. Sah, and R. Ptucha, "General-purpose deep point cloud feature extractor," in *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, March 2018.

[67] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph CNN for learning on point clouds," *ACM Transactions on Graphics (TOG)*, 2019.

[68] H. Lei, N. Akhtar, and A. Mian, "Spherical kernel for efficient graph convolution on 3d point clouds," *IEEE TPAMI*, 2020.

[69] B.-S. Hua, M.-K. Tran, and S.-K. Yeung, "Pointwise convolutional neural networks," in *CVPR*, Jun. 2018.

[70] S. Xie, S. Liu, Z. Chen, and Z. Tu, "Attentional ShapeContextNet for point cloud recognition," in *CVPR*, Jun. 2018.

[71] A. Komarichev, Z. Zhong, and J. Hua, "A-CNN: Annularly convolutional neural networks on point clouds," in *CVPR*, June 2019.

[72] M. Atzmon, H. Maron, and Y. Lipman, "Point convolutional neural networks by extension operators," *ACM Transactions on Graphics (TOG)*, vol. 37, no. 4, 2018.

[73] Y. Shen, C. Feng, Y. Yang, and D. Tian, "Mining point cloud local structures by kernel correlation and graph pooling," in *CVPR*, Jun. 2018.

[74] H. Thomas, C. R. Qi, J.-E. Deschaud, B. Marcotegui, F. Goulette, and L. J. Guibas, "KPConv: Flexible and deformable convolution for point clouds," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019.

[75] W. Wu, Z. Qi, and L. Fuxin, "PointConv: Deep convolutional networks on 3D point clouds," in *CVPR*, 2019.

[76] Y. Xu, T. Fan, M. Xu, L. Zeng, and Y. Qiao, "SpiderCNN: Deep learning on point sets with parameterized convolutional filters," in *Computer Vision – ECCV*, 2018, vol. 11212.

[77] Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen, "PointCNN: Convolution on X-transformed points," in *NIPS*. Curran Associates, Inc., 2018.

[78] F. Groh, P. Wieschollek, and H. P. A. Lensch, "Flex-Convolution: Million-scale point-cloud learning beyond grid-worlds," in *Computer Vision – ACCV 2018*. Springer International Publishing, 2019, vol. 11361.

[79] Y. Liu, B. Fan, S. Xiang, and C. Pan, "Relation-shape convolutional neural network for point cloud analysis," in *CVPR*, 2019.

[80] X. Liu, Z. Han, Y.-S. Liu, and M. Zwicker, "Point2Sequence: Learning the shape representation of 3D point clouds with an attention-based sequence to sequence network," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019.

[81] J. Lee, Y. Lee, J. Kim, A. Kosiorek, S. Choi, and Y. W. Teh, "Set transformer: A framework for attention-based permutation-invariant neural networks," in *ICML*, 2019.

[82] M. Gadelha, R. Wang, and S. Maji, "Multiresolution tree networks for 3D point cloud processing," in *ECCV*, 2018.

[83] P. Achlioptas, O. Diamanti, I. Mitliagkas, and L. Guibas, "Learning representations and generative models for 3D point clouds," in *International Conference on Machine Learning*, 2018.

[84] Y. Yang, C. Feng, Y. Shen, and D. Tian, "FoldingNet: Point cloud auto-encoder via deep grid deformation," in *CVPR*, 2018.

[85] S. Xie, J. Gu, D. Guo, C. R. Qi, L. Guibas, and O. Litany, "Pointcontrast: Unsupervised pre-training for 3d point cloud understanding," in *ECCV*, 2020.

[86] J. Masci, D. Boscaini, M. Bronstein, and P. Vandergheynst, "Geodesic convolutional neural networks on riemannian manifolds," in *Proceedings of the IEEE*

international conference on computer vision workshops, 2015.

[87] D. Boscaini, J. Masci, E. Rodolà, and M. Bronstein, "Learning shape correspondence with anisotropic convolutional neural networks," in *NIPS*, 2016.

[88] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein, "Geometric deep learning on graphs and manifolds using mixture model CNNs," in *CVPR*, 2017.

[89] J. Bruna, W. Zaremba, A. Szlam, and Y. Lecun, "Spectral networks and locally connected networks on graphs," in *International Conference on Learning Representations (ICLR2014), CBLS, April 2014*, 2014.

[90] J. Atwood and D. Towsley, "Diffusion-convolutional neural networks," in *NIPS*, 2016.

[91] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv:1609.02907*, 2016.

[92] M. Dominguez, F. P. Such, S. Sah, and R. Ptucha, "Towards 3D convolutional neural networks with meshes," in *2017 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2017.

[93] N. Verma, E. Boyer, and J. Verbeek, "Feastnet: Feature-steered graph convolutions for 3d shape analysis," in *CVPR*, 2018.

[94] H. Xu, M. Dong, and Z. Zhong, "Directionally convolutional networks for 3D shape segmentation," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017.

[95] I. Kostrikov, Z. Jiang, D. Panozzo, D. Zorin, and J. Bruna, "Surface networks," in *CVPR*, 2018.

[96] Y. Feng, Y. Feng, H. You, X. Zhao, and Y. Gao, "MeshNet: mesh neural network for 3D shape representation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019.

[97] R. Hanocka, A. Hertz, N. Fish, R. Giryes, S. Fleishman, and D. Cohen-Or, "MeshCNN: a network with an edge," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 4, 2019.

[98] V. Hegde and R. Zadeh, "FusionNet: 3D object classification using multiple data representations," *CoRR*, vol. abs/1607.05695, 2016.

[99] H. You, Y. Feng, R. Ji, and Y. Gao, "PVNet: A joint convolutional network of point cloud and multi-view for 3D shape recognition," in *Proceedings of the 26th ACM International Conference on Multimedia*, ser. MM '18. Association for Computing Machinery, 2018.

[100] D. Maturana, "3D/Volumetric Convolutional Neural Networks with Theano+Lasagne," 2015. [Online]. Available: https://github.com/dimatura/voxnet

[101] Z. Wu, "3D ShapeNets implementation in Matlab," 2005. [Online]. Available: https://github.com/zhirongw/3DShapeNets

[102] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv:1502.03167*, Feb. 2015.

[103] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, Inception-ResNet and the impact of residual connections on learning," *arXiv:1602.07261*, Feb. 2016.

[104] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Weinberger, "Deep networks with stochastic depth," *arXiv:1603.09382*, Mar. 2016.

[105] A. Brock, "VRN in Theano with Lasagne," 2016. [Online]. Available: https://github.com/ajbrock/Generative-and-Discriminative-Voxel-Modeling

[106] N. Sedaghat, M. Zolfaghari, E. Amiri, and T. Brox, "ORION in Caffe," 2016. [Online]. Available: https://github.com/lmb-freiburg/orion

[107] X. Xu, "Beam search for learning a deep convolutional neural network of 3D shapes - Matlab implementation," 2018. [Online]. Available: https://github.com/xuxucmkox/3D-shape-Classification-Beam-Search

[108] D. Meagher, "Octree encoding: A new technique for the representation, manipulation and display of arbitrary 3-D objects by computer," Image Processing Laboratory, Rensselaer Polytechnic Institute, Tech. Rep., Oct. 1980.

[109] G. Riegler and J. Lee, "OctNet implementation in Torch," 2017. [Online]. Available: https://github.com/griegler/octnet

[110] P.-S. Wang, Y. Liu, Y.-X. Guo, C.-Y. Sun, and X. Tong, "O-CNN in Caffe," 2018. [Online]. Available: https://github.com/Microsoft/O-CNN

[111] A. Sharma, "Source code for 3D volumetric denoising auto-encoder," 2017. [Online]. Available: https://github.com/Not-IITian/VCONV-DAE

[112] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *NIPS*, 2014.

[113] C. Zhang, "3D generative adversarial network Torch implementation," 2016. [Online]. Available: https://github.com/zck119/3dgan-release

[114] S. Mou and E. Zheng, "Keras implementation of 3D-ED-GAN," 2018. [Online]. Available: https://github.com/Fdevmsy/3D_shape_inpainting

[115] S. Liu and A. Ororbia, "Variational shape learner TensorFlow implementation," 2017. [Online]. Available: https://github.com/lorenmt/vsl

[116] Y. Li, "FPNN implementation in Caffe," 2016. [Online]. Available: https://github.com/yangyanli/FPNN

[117] T. Le, "PointGrid implementation in TensorFlow," 2018. [Online]. Available: https://github.com/trucleduc/PointGrid

[118] S. Hang, S. Maji, Vangelis, J. Hackenberg, haibin, and A. RoyChowdhury, "Multi-view CNN (MVCNN) shape recognition in Matlab," 2014. [Online]. Available: https://github.com/suhangpro/mvcnn

[119] J.-C. Su, M. Gadelha, and R. Wang, "Multi-view CNN in Pytorch," 2018. [Online]. Available: https://github.com/jongchyisu/mvcnn_pytorch

[120] K. Sarkar, "MVCNN for multi-layered height-maps," 2018. [Online]. Available: https://github.com/krips89/mlh_mvcnn

[121] S. Kim, "GVCNN in TensorFlow," 2018. [Online]. Available: https://github.com/ace19-dev/gvcnn-tf

[122] waxnkw, "GVCNN - PyTorch Version," 2019. [Online]. Available: https://github.com/waxnkw/gvcnn-pytorch

[123] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Ben-

gio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *arXiv:1406.1078*, Jun. 2014.
[124] H. Zhizhong, S. Mingyang, and L. Zhenbao, "SeqViews2SeqLabels in Tensorflow," 2018. [Online]. Available: https://github.com/mingyangShang/SeqViews2SeqLabels
[125] A. Kanezaki, "RotationNet in Caffe," 2018. [Online]. Available: https://github.com/kanezaki/rotationnet
[126] ——, "RotationNet in PyTorch," 2018. [Online]. Available: https://github.com/kanezaki/pytorch-rotationnet
[127] A. A. Soltani, "Synthesizing 3D shapes via modeling multi-view depth maps and silhouettes with deep generative networks in Torch," 2017. [Online]. Available: https://github.com/Amir-Arsalan/Synthesize3DviaDepthOrSil
[128] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *NIPS*. Curran Associates, Inc., 2015.
[129] A. Sinha, "learning_geometry_images implementation," 2016. [Online]. Available: https://github.com/sinhayan/learning_geometry_images
[130] B. Shi, "DeepPano panorama projection code," 2017. [Online]. Available: https://github.com/bgshih/deeppano
[131] C. f. D. G. U. o. P. Esteves, "Demo code for the paper learning SO(3) equivariant representations with spherical CNNs," 2017. [Online]. Available: https://github.com/daniilidis-group/spherical-cnn
[132] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet in TensorFlow," 2016. [Online]. Available: https://github.com/charlesq34/pointnet
[133] C. Choy, J. Gwak, and S. Savarese, "4d spatio-temporal convnets: Minkowski convolutional neural networks," in *CVPR*, 2019.
[134] M. Zaheer, "Deep Sets PyTorch implementation," 2017. [Online]. Available: https://github.com/manzilzaheer/DeepSets
[135] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet++ in TensorFlow," 2017. [Online]. Available: https://github.com/charlesq34/pointnet2
[136] J. Li, "SO-Net in TensorFlow," 2018. [Online]. Available: https://github.com/lijx10/SO-Net
[137] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, Sep. 1975.
[138] R. Klokov and V. Lempitsky, "KD-Net in Theano and Lasagne," 2017. [Online]. Available: https://github.com/Regenerator/kdnets
[139] M. Dominguez, "G3DNet in Tensorflow," 2018. [Online]. Available: https://github.com/WDot/G3DNet
[140] Y. Wang, Y. Sun, and Z. Liu, "DGCNN implementation in TensorFlow and PyTorch," 2018. [Online]. Available: https://github.com/WangYueFt/dgcnn
[141] H. Lei, "Sph3d-gcn TensorFlow implementaion," 2020. [Online]. Available: https://github.com/hlei-ziyan/SPH3D-GCN
[142] B.-S. Hua and Q.-H. Pham, "Code for pointwise convolutional neural networks," 2018. [Online]. Available: https://github.com/hkust-vgd/pointwise
[143] A. Komarichev, "A-CNN TensorFlow implementaion," 2019. [Online]. Available: https://github.com/artemkomarichev/a-cnn
[144] M. Atzmon, "PCNN TensorFlow implementaion," 2018. [Online]. Available: https://github.com/matanatz/pcnn
[145] C. Feng, "KCNet implementation," 2018. [Online]. Available: http://www.merl.com/research/license#KCNet
[146] H. Thomas, "KPConv implementation," 2019. [Online]. Available: https://github.com/HuguesTHOMAS/KPConv
[147] W. Wu, "PyTorch version of PointConv," 2019. [Online]. Available: https://github.com/DylanWusee/pointconv_pytorch
[148] Y. Xu, "SpiderCNN TensorFlow implementation," 2018. [Online]. Available: https://github.com/xyf513/SpiderCNN
[149] Y. Li, R. Bu, G. Smith, M. C. Sun, and T. Schattschneider, "PointCNN TensorFlow implementation," 2017. [Online]. Available: https://github.com/yangyanli/PointCNN
[150] P. Wieschollek, F. Groh, and M. Clark, "Flex-Convolution TensorFlow implementation," 2018, note: classifier details discussed in Issue #13. [Online]. Available: https://github.com/cgtuebingen/Flex-Convolution
[151] Y. Liu and M. Zahran, "RS-CNN implementation," 2019. [Online]. Available: https://github.com/Yochengliu/Relation-Shape-CNN
[152] D. Hu, "An introductory survey on attention mechanisms in NLP problems," in *Proceedings of SAI Intelligent Systems Conference*. Springer, 2019.
[153] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, 1997.
[154] X. Liu and B. Kakillioglu, "Point2Sequence TensorFlow implementation," 2019. [Online]. Available: https://github.com/liuxinhai/Point2Sequence
[155] J. Lee, "Set transformer PyTorch implementation," 2019. [Online]. Available: https://github.com/juho-lee/set_transformer
[156] M. Gadelha, "MRTNet PyTorch implementation," 2018. [Online]. Available: https://github.com/matheusgadelha/MRTNet
[157] P. Achlioptas and O. Diamanti, "Auto-encoding & generating 3D point-clouds," 2017. [Online]. Available: https://github.com/optas/latent_3d_points
[158] Y. Yang, C. Feng, Y. Shen, and D. Tian, "FoldingNet code request form," 2018. [Online]. Available: http://www.merl.com/research/?research=license-request&sw=FoldingNet
[159] S. Xie, L. Yuecheng, and J. Hou, "Pointcontrast implementation in PyTorch," 2020. [Online]. Available: https://github.com/facebookresearch/PointContrast
[160] J. Masci, "ShapeNet data preparation toolbox," 2015. [Online]. Available: https://github.com/jonathanmasci/ShapeNet_data_preparation_toolbox
[161] D. Boscaini, "Anisotropic CNN Matlab and Theano implementation," 2017. [Online]. Available: https://github.com/davideboscaini/acnn

[162] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein, "MoNet Theono and Lasagne, TensorFlow and Matlab implementations," 2017. [Online]. Available: http://geometricdeeplearning.com/code/MoNet/MoNet_code.tar.gz
[163] F. R. Chung and F. C. Graham, *Spectral graph theory*. American Mathematical Soc., 1997, no. 92.
[164] M. Defferrard and D. P. Larson, "Convolutional neural networks on graphs with fast localized spectral filtering," 2015. [Online]. Available: https://github.com/mdeff/cnn_graph
[165] J. Atwood, "An implementation of diffusion-convolutional neural networks in Lasagne and Theano," 2017. [Online]. Available: https://github.com/jcatw/dcnn
[166] T. N. Kipf, T. Kukurin, YaGuang, C. Pearson, B. Camargo, L. (alphadl), S. Schmidt, and B. Raghunathan, "Implementation of graph convolutional networks in TensorFlow," 2016. [Online]. Available: https://github.com/tkipf/gcn
[167] N. Verma, "Feastnet TensorFlow implementaion," 2018. [Online]. Available: https://github.com/nitika-verma/FeaStNet
[168] Z. Jiang, "Surface networks PyTorch implementaion," 2018. [Online]. Available: https://github.com/jiangzhongshi/SurfaceNetworks
[169] Y. Feng, "MeshNet PyTorch implementaion," 2018. [Online]. Available: https://github.com/iMoonLab/MeshNet
[170] Z. Lian, A. Godil, B. Bustos, M. Daoudi, J. Hermans, S. Kawamura, Y. Kurita, G. Lavoua, and P. Dp Suetens, "Shape retrieval on non-rigid 3D watertight meshes," in *Eurographics workshop on 3D object retrieval (3DOR)*. Citeseer, 2011.
[171] R. Hanocka, J. Carnero, J. Zheng, and Kailash2906, "MeshCNN PyTorch implementaion," 2019. [Online]. Available: https://github.com/ranahanocka/MeshCNN
[172] R. Polikar, "Ensemble learning," in *Ensemble machine learning*. Springer, 2012.
[173] M. Person, M. Jensen, A. O. Smith, and H. Gutierrez, "Multimodal Fusion Object Detection System for Autonomous Vehicles," *Journal of Dynamic Systems, Measurement, and Control*, vol. 141, no. 7, 05 2019.
[174] D. Koguciuk and Ł. Chechliński, "Point cloud ensemble," 2018. [Online]. Available: https://github.com/dkoguciuk/ensemble_learning_for_point_clouds
[175] H. You and Y. Feng, "PVRNet (PVNet implementation)," 2018. [Online]. Available: https://github.com/Hxyou/PVRNet
[176] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu, "ShapeNet: An information-rich 3D model repository," Stanford University — Princeton University — Toyota Technological Institute at Chicago, Tech. Rep. arXiv:1512.03012, 2015.
[177] K. Museth, J. Lait, J. Johanson, J. Budsberg, R. Henderson, M. Alden, P. Cucka, D. Hill, and A. Pearce, "OpenVDB: An open-source data structure and toolkit for high-resolution volumes," in *ACM SIGGRAPH 2013 Courses*. ACM, 2013.
[178] S. Lloyd, "Least squares quantization in PCM," *IEEE Transactions on Information Theory*, vol. 28, no. 2, Mar. 1982.
[179] F. Williams, "Point cloud utils - a python library for common tasks on 3D point clouds," 2019. [Online]. Available: https://github.com/fwilliams/point-cloud-utils
[180] D. Merkel, "Docker: Lightweight linux containers for consistent development and deployment," *Linux J.*, vol. 2014, no. 239, Mar. 2014.
[181] C. Yeung, "VGG19 and VGG16 on Tensorflow," 2017. [Online]. Available: https://github.com/machrisaa/tensorflow-vgg

Martin Mirbauer received the master's degree in computer graphics from Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic in 2018; currently a Ph.D. student there. His main research includes machine learning for 3D content creation.

Miroslav Krabec received the master's degree in artificial intelligence from Faculty of Mathematics and Physics, Charles University, Prague, in 2019.

Jaroslav Křivánek received the Ph.D. degree from INRIA, Rennes. He was an associate professor in computer science with Charles University, Prague, and director of research with Chaos Czech a.s.. His research interests included realistic rendering with the focus on Monte Carlo methods for light transport simulation, extending to tools for content creation.

Elena Šikudová received the Ph.D. degree in geometry and topology from Faculty of Mathematics, Physics and Informatics at Comenius University, Bratislava. She is an associate professor in computer science with Charles University, Prague. Her research interests include computer vision, HDR imaging, visual perception, and image processing.