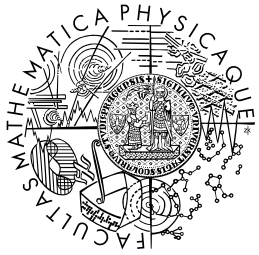


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Bc. Miroslav Krabec

3D object classification using neural networks

Department of Software and Computer Science Education

Supervisor of the master thesis:	doc. Ing. Jaroslav Křivánek, Ph.D.
Study programme:	Computer Science
Study branch:	Artificial Intelligence

Prague 2019

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ..... signature of the author

Dedication.

Title: 3D object classification using neural networks

Author: Bc. Miroslav Krabec

Department: Department of Software and Computer Science Education

Supervisor: doc. Ing. Jaroslav Křivánek, Ph.D., Department of Software and Computer Science Education

Abstract: Abstract.

Keywords: deep learning, classification, neural networks 3D

Contents

1. Introduction

In this chapter we provide reasons and motivation for working on the problem of 3D classification, define the problem and give a quick outline of the thesis.

1.1 Motivation and Goals

Recognition and generation of 3D shapes is quickly becoming one of the widely researched topic in the field of artificial intelligence. It can be applied in a vast number of fields such as driving of autonomous cars, analysis of medical scans as well as various fields of computer graphics. We approach this problem from the standpoint of computer graphics as we are interested in developing tools for content creators, architects or interior designers. In order to develop such tools it is necessary to start with the simplest problem which is classification. There are many more or less successful approaches to 3D classification, most of them employing some kind of deep artificial neural network. However their relative performance has never been objectively evaluated.

The goal of this thesis is therefore to test existing techniques for 3D classification, find out how difficult is to replicate their reported results and finally to compare them and evaluate their applicability in real-world setting.

1.2 Problem Statement

As we intend to develop tools for computer graphics, our input is in the form of a 3D mesh. A 3D mesh is usually supplied as list of vertices – triplets of coordinates in euclidean space and list of faces – usually three vertices each forming a triangle. 3D mesh files can contain other information such as texture coordinates and materials, but we will be ignoring these. Our goal is to classify mesh files to several given categories (such as car, chair, sofa, etc.) using methods of supervised learning.

We can define the problem formally as follows: we are given a set of training examples $\{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$ where, in our case, x_i is a 3D shape representation and y_i is a numerical encoding of corresponding label. Each shape belongs to exactly one class. The goal of classification can be formulated as learning a parametric model $P : X \to Y$ where X is a space of possible 3D shapes and Y is a space of labels. This model should be able to predict the correct class label for each model from X. As the mesh format is not suitable for direct use with neural networks, we have to be able to convert meshes to other representations: point clouds, volumes, or images.

1.3 Scope of the Thesis

We limit the scope of this thesis in the following ways. We will perform classification only on aligned 3D shapes, as not all networks can easily cope with arbitrary rotations. We will only consider simple classification, although most of the networks can be extended to perform part segmentation as well as new shape

generation. We will not test any large ensembles of networks: although they produce better results, they are usually big and cumbersome to use, while achieving only marginal improvements. We also consider only networks with publicly available code as implementing all the different techniques is far beyond the scope of this work.

1.4 Thesis Outline

In this chapter 1 we presented basics of the problem as well as our motivation for this work. In chapter 2 we provide a brief introduction to artificial neural networks. In chapter 3 we introduce different approaches to 3D classification and their implementations. In chapter 4 we discuss chosen methods, describe datasets, procedures of data conversion and technical setup of our framework. In chapter 5 we present the setup and results of our experiments, comparison of tested networks and analysis of the dataset. In the final chapter 6 we summarize our findings and suggest directions for future work.

2. Theoretical Background

This chapter contains a general overview of artificial neural networks in general as well as a quick description of software frameworks for machine learning.

2.1 Artificial Neural Networks

Artificial neural networks are widely successful in a great number of areas of computer science, often achieving better than human efficiency. This chapter offers a brief introduction to the principles of artificial neural networks. For more in depth information we recommend Goodfellow et al. [2016].

2.1.1 Feedforward Neural Networks

Artificial neural networks are computing systems inspired by the structure of central nervous system of animals. A basic computing unit of the network is called *neuron*, which performs some simple computation on its inputs and produces its output. Neurons are connected by weighted connections and so create a network.

Artificial neural networks are parametric models – parameters are usually called *weights* and are learned during the training of the network. On the other hand *hyperparameters* are parameters whose values are set before the learning process begins. Artificial neural networks can be trained for a variety of tasks by minimizing some *loss function*. In the case of classification a *cross-entropy* loss is commonly used (Goodfellow et al. [2016]). The networks are trained by some variant of gradient descent algorithm – backpropagating the error of the trained task through the network in direction from outputs to inputs.

Feedforward Neural Networks are networks where the information is only passed in one direction, so the graph of the network is an acyclic directed graph. The simplest network architecture is the so called *Single-layer perceptron*. It consists of a single layer of output neurons; the inputs are fed directly to the outputs via a series of weights. Each neuron can also have a bias value and activation function. Then the output of a neuron is computed:

$$output_j = f(\sum_i w_{ij} \times inputs_i + b_j)$$

Where f is an activation function, w_{ij} is a weight of the connection from i-th input neuron to j-th output neuron and b_j is its bias value. Figure 2.1 shows a diagram of a single layered perceptron.

The activation function can be an arbitrary function but is required to be non-linear and differentiable. Most commonly used are the sigmoid function, hyperbolic tangent and rectified linear unit (ReLU), defined as $ReLU(x) = max(0, x)$. ReLU is not a differentiable function in $x = 0$, but in practice this usually does not occur and software frameworks implementing artificial neural networks handle this by returning one of the one-sided derivatives.

We can insert one or more layers of neurons between input and output obtaining a *Multi-layered perceptron*. A neuron from layer k gets its input from all neurons from layer $(k - 1)$ and passes its output to all neurons in the layer

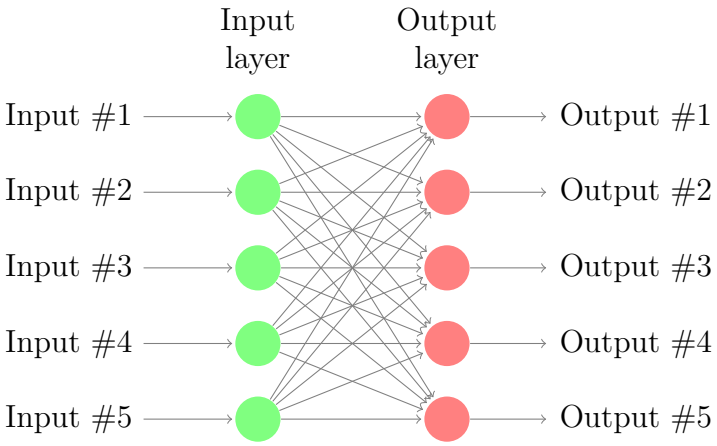


Figure 2.1: Single-layer perceptron

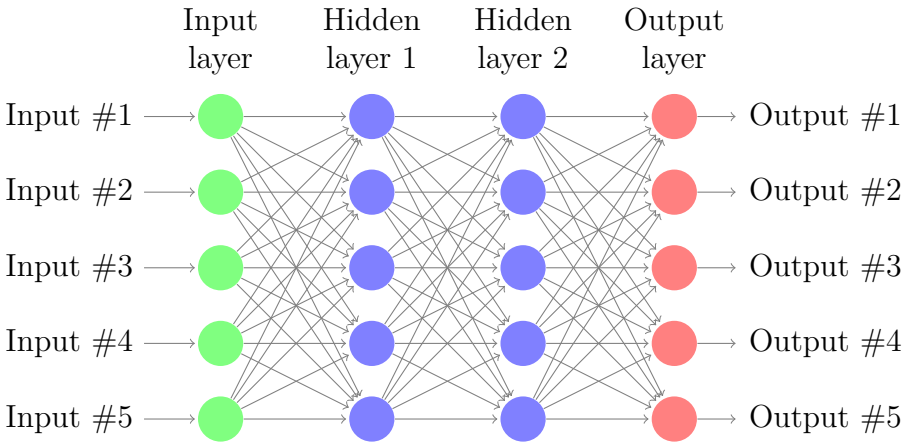


Figure 2.2: Multi-layer perceptron

$(k+1)$. Therefore this type of layer is called *fully connected layer* or *dense layer*. We can also rewrite all the weights to a matrix form and obtain the so called *weight matrix*. For each dense layer we get one matrix, where W_{ij} is weight of the connection from neuron i to neuron j. Using the matrix notation, output of a fully connected layer k can be expressed as follows:

$$output_k = f(W_k \times output_{k-1} + b_k)$$

Where f is an activation function, W_k is the weight matrix for layer k and b_k is a corresponding vector of biases.

Multi layered perceptrons, despite being old Rosenblatt [1961], still create the basis in modern systems as fully connected layers are used in almost all other types of neural networks especially as last layers in the network, producing feature vectors or classification distributions. Figure 2.2 shows a diagram of a multi layered perceptron.

2.1.2 Convolutional Neural Networks

Convolutional Neural Networks, first introduced in LeCun et al. [1989], are designed to capture the spatio-temporal data better than the standard fully connected layers. They are most successful in processing two-dimensional images so

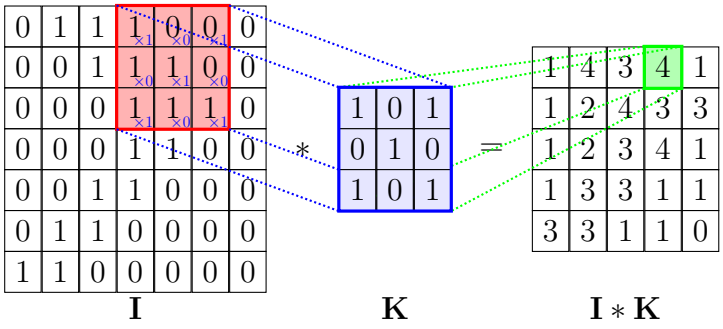


Figure 2.3: Illustration of 2D convolution with one 3x3 filter and valid padding.

we will describe this case. However, also one-dimensional and three dimensional convolutions are used as shall be described later.

Weights of the convolutional layers are connected to only a small region of the data and shared across the spatio-temporal dimensions. In the case of images this means that the convolutional layer is connected to only small patches of the image and weights are shared among these patches. Weights of the convolutional layers are typically called *filters* or *kernels*. A common approach is to slide the filter across the whole image, computing local features for each pixel. Two dimensional convolution can be defined:

$$Conv(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n)$$

where I is the input image and K represents the weights of the kernel. Figure 2.3 shows an illustration of 2D convolution.

The speed of the sliding window is a parameter called *stride*. By having a stride bigger than one we skip some pixels in the input, and obtain an image with smaller width and height.

We also need to employ one of the padding schemes on the edges of input image. The most usual type of padding is *zero padding*, which counts areas outside of picture as having value of zero and is preserving original dimensions. Another approach, *valid padding*, is to ignore the edge pixels of the input image altogether, sliding the filter only across valid positions – this produces a smaller output image. By stacking more convolutional layers atop each other and creating a deep convolutional network, global features of the image can be extracted.

Another important type of layer used in convolutional neural networks is a *pooling layer*. It is usually used on the feature maps obtained by convolutional layers. The goal of the pooling is to get some translation invariance in produced features. Similarly to the convolution, pooling also scans the entire input feature map in a sliding-window fashion. However it does not perform convolution but maximum, average or a similar function. Unlike with convolution, stride bigger than one is used when using pooling in order to reduce the size of the output features. The most commonly used type of pooling – max pooling – can be described by the following formula:

$$maxpool(I)[x,y] = \max_{\substack{0 \le i < d_h \\ 0 \le j < d_w}} I[x + i - \lfloor \frac{d_h}{2} \rfloor, y + j - \lfloor \frac{d_w}{2} \rfloor]$$

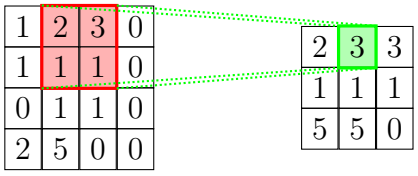


Figure 2.4: Illustration of 2D maximum pooling with valid padding.

Where x and y are coordinates of a pixel in the picture, d_h and d_w are sizes of the sliding window. Figure 2.4 shows a diagram illustrating maximum pooling in two dimensions.

Convolutional layers are much more efficient than fully connected layers as they share their parameters across the image and have been very successful in a variety of image, video, and natural language processing tasks. This can be transferred to our 3D classification task as several networks use rendered images of 3D models, employing 2D convolutions, with great success.

2.1.3 Recurrent Neural Networks

Another widely used class of artificial neural networks are *recurrent neural networks* (RNN). In contrast to the previously described networks, they take as their input also their previous state representing a kind of memory. Recurrent neural networks are well suited for processing of sequenced data, such as video, text or speech. A typical architecture of a RNN is the *encoder-decoder* architecture. Encoder produces a feature vector by processing the input sequence. Decoder then constructs an output sequence from the feature vector. To give the network control over its memory, two types of cells were devised: *long short-term memory unit* (LSTM) Hochreiter and Schmidhuber [1997] and *gated recurrent unit* (GRU) Cho et al. [2014]. An important concept, which leads to better performance, is *attention* Bahdanau et al. [2014] – it allows the network to learn which parts of the input sequence are important and how they correspond to the output sequence.

Only one of the networks tested uses the RNN architecture and techniques described in this section, and so we refer to Goodfellow et al. [2016] for further information.

2.1.4 Regularization

Overfitting is a common problem in machine learning. It is a phenomenon when a model represents the training set too well and fails to generalize. To avoid this problem several regularization techniques are employed. We can add regularization directly to the optimized loss function. This is usually implemented by adding some term which keeps the weights of the network in low absolute values.

Dropout Srivastava et al. [2014] is a stochastic regularization technique; during training, at every iteration, it randomly selects some nodes and removes them along with all of their incoming and outgoing connections. So each iteration has a different set of nodes and this results in a different set of outputs. It is a

very effective technique which can be thought of as training a whole ensemble of networks at once. Dropout is usually applied after each fully connected layer.

Another common way how to avoid overfitting is by increasing the size of the training set by *data augmentation*. Specifics of data augmentation depend on the kind of data we are working with, but in general the dataset is increased by creating new instances from the training dataset. For example with image data, we can use geometric transformations such as mirroring, rotating, translating or scaling.

These techniques are standard when training neural network and are employed in one form or another in all the networks we tested.

2.1.5 Training

In recent years very deep (tens of layers) neural networks are used. To train such a network several improvements were developed. Mini-batch training is used almost in all cases. The training dataset is divided into small chunks (typically 32 or 64 examples) and one batch is presented, gradient is computed and weights are updated correspondingly in each step of the training. This is much more efficient than computing the gradient of the whole dataset and has some positive regularization effects.

For deep networks it is no longer sufficient to use a simple gradient descent algorithm. To speed up and stabilize the training process, algorithms with momentum, such as Nesterov momentum Sutskever et al. [2013], have been used. An important hyperparameter of the network is the learning rate, which controls the speed of training. Set the learning rate too small and the network can not learn at all, too big and the weights can diverge. To solve this problem, algorithms with adaptive learning rates, such as RMSProp Hinton et al. or ADAM Kingma and Ba [2014], are used.

Another obstacle in training is that the information contained in the gradient gets lost in deep neural networks during the backpropagation phase of the algorithm. It either diminishes to zero or grows rapidly and diverges. This can be avoided by a good choice of the activation function (in modern networks mainly ReLU is used) or by some normalization technique. A prime example of this is *batch normalization* Ioffe and Szegedy [2015], which normalizes the inputs of the layer by subtracting the mean of the batch and dividing by its standard deviation. These changes would be discarded by the learning algorithm, so we add two learnable parameters representing the mean and standard deviation, and the output of the layer is again denormalized using these parameters. This effectively allows the network to learn the correct scaling of the weights using only two parameters instead of changing the whole network, which leads to a much greater stability of the training.

Another widely used technique allowing training of very deep networks are *residual connections* Szegedy et al. [2016]. A residual connection allows the network to skip the layer and choose to work with the input instead. This makes copying the information through the network possible and helps reduce the vanishing gradient problem.

The above described techniques are used in several neural networks for 3D classification, as some of them use very deep 2D or 3D convolutional networks.

2.2 Deep Learning Frameworks

In recent years several software frameworks for machine learning in general and for deep learning in particular have been developed. They are usually implemented in C++ for performance but provide a Python API for more convenient use. All of the following libraries are open-source and publicly available. They also implement support for running the machine learning algorithms on the GPU – a key feature for fast training of deep neural networks.

One of the most widely used deep learning frameworks is TensorFlow Martín Abadi et al. [2015] developed by a team at Google. It is a symbolic math library and implements all the standard neural layers as well as many of the latest developments in the area of deep learning.

PyTorch Paszke et al. [2017] is a Python extension of a Torch machine learning library. It is primarily developed by Facebook. It focuses on simple usage and Python integration and implements all the standard functions as well as extended tools for various areas of machine learning.

Caffe Jia et al. [2014] is a deep learning framework originally developed at University of California, Berkeley. It offers good speed and training models without the need to write any code, just network definitions have to be provided. It does not seem to be developing as quickly as the aforementioned frameworks. There was also Caffe2 developed by Facebook, but it was merged into PyTorch.

Theano Theano Development Team [2016] is a Python library for manipulating and evaluating mathematical expressions, primarily developed by the Montreal Institute for Learning Algorithms (MILA) at the Université de Montréal. Lasagne Dieleman et al. [2015] is a lightweight library which uses Theano for machine learning computations. It also does not seem to be developing quickly enough at present.

Other favourite deep learning frameworks, which we do not use in our work, include Microsoft Cognitive Toolkit (CNTK) Seide and Agarwal [2016] developed by Microsoft and Keras Chollet and others [2015] which is a high-level API focusing on being user friendly and uses TensorFlow as its backend, but is modifiable to work with other frameworks as well.

3. Survey of 3D Classification Methods

In recent years many techniques for classifying 3D shapes by means of artificial neural networks have been devised. In this chapter we present most of the commonly used and successful of them. As mentioned in previous chapters the usual mesh format is not suitable for processing by a neural network directly so we divide the networks according to the format they use as their input: voxel-based, multi-view, and point-cloud based. Table 3.1 shows a list of neural networks described in this chapter.

3.1 Voxel Based Neural Networks

As 2D convolutional neural networks were a great breakthrough in image recognition, it is natural to try to generalize this approach to three dimensions. Instead of pixels we use 3D occupancy grid of so called voxels. As shown later we can easily extend convolutions to three dimensions. Convolutions seem to be suitable for the task as they can make use of the spatial structure of the problem. However they are computationally demanding and voxel grids have high memory requirements as their size grows with the cube of the resolution. For this reason only relatively small resolutions can be used, the most usual being 32^3.

3.1.1 VoxNet

First of the successful systems applying 3D convolutions to occupancy grids is VoxNet Maturana and Scherer [2015], which we will use as an example of a network using a shallow convolutional architecture. In VoxNet, the occupancy grid is processed by 3D convolutions which extract local features and lower the resolution. The convolution result is then passed to a ReLU layer to achieve nonlinearity. Maximum pooling is then performed in order to get better representation and to further lower the number of parameters needed. Finally the occupancy grid is flattened and passed to a fully connected layer which outputs resulting feature vector. Figure 3.1 shows a diagram of the VoxNet architecture.

As is common with neural networks, data augmentation is a very important part of the training process. VoxNet uses rotation along the vertical axis as its main augmentation technique. During training it creates n copies of each input instance each 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.

Results of VoxNet were improved by ORION Sedaghat et al. [2016], wherein the classification task was augmented with an orientation estimation task and FusionNet Hegde and Zadeh [2016] combining 3D convolutions on voxel representation with a multi-view approach.

Acronym	Reference	Framework	Included
Voxel			
VoxNet	Maturana and Scherer [2015]	TensorFlow	No
ORION	Sedaghat et al. [2016]	Caffe	No
FusionNet	Hegde and Zadeh [2016]	Not available	No
VRN	Brock et al. [2016]	Lasagne	Yes
O-CNN	Wang et al. [2017]	Caffe	Yes
AO-CNN	Wang et al. [2018]	Caffe	Yes
Multi-view			
VGG-voting	Simonyan and Zisserman [2014]	TensorFlow	Yes
MVCNN	Su et al. [2015]	TensorFlow	Yes
MVCNN2	Su et al. [2018]	PyTorch	
RotationNet	Kanezaki et al. [2016]	Caffe	Yes
Seq2Seq	Zhizhong et al. [2018]	TensorFlow	Yes
Point cloud			Yes
PointNet	Qi et al. [2016]	TensorFlow	Yes
PointNet++	Qi et al. [2017]	TensorFlow	Yes
SO-Net	Li et al. [2018]	PyTorch	Yes
KD-Net	Klokov and Lempitsky [2017]	Lasagne	Yes
GraphNet	Dominguez et al. [2018]	TensorFlow	No

Table 3.1: List of examined neural networks. There is a citation of an original paper, a framework of the publicly available code and if we included the network in our testing.

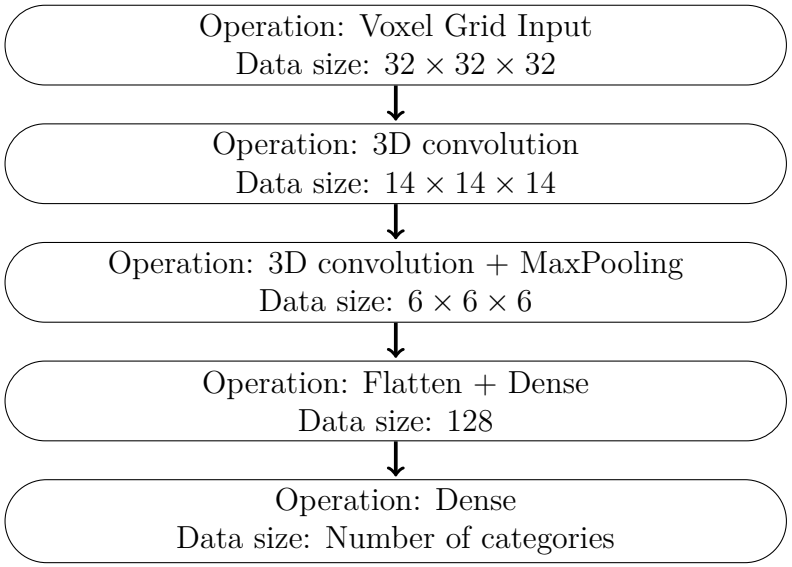


Figure 3.1: VoxNet architecture

3.1.2 Voxception Residual Network

Voxception Residual Network Brock et al. [2016] is inspired by deep residual convolutional networks for image recognition which are the state of the art approach for this task. It uses Inception-style Szegedy et al. [2016] modules, batch normalization (subsection 2.1.4), residual connections (subsection 2.1.5) and stochastic network depth Huang et al. [2016]. The Voxception network consists of several sequential voxception modules. These modules should enable for information to propagate through the network through many possible "pathways", while still maintaining simplicity and efficiency.

For example, one of the basic blocks concatenates the result of a $3 \times 3 \times 3$ convolution and the result of a $1 \times 1 \times 1$ convolution, so the network can learn which of these filters to apply. A diagram of this block with added residual connection is in Figure 3.2. There are several types of the so called Voxception blocks with residual connections with pre-activation (nonlinearity is used before the addition) employed in the network.

The best performing architecture consists of voxception blocks as well as downsampling blocks which enable the network to choose the best downsampling methods. The deepest path through the network is 45 layers, and the shallowest path (assuming all droppable non-residual paths are dropped) is 8 layers deep. The model is quite big and slow to train, authors report one epoch taking around six hours on a single Titan X GPU, which is in line with our results. 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 binary voxel representation but with binary voxel range $\{-1, 5\}$ instead $\{0, 1\}$ to encourage the network to pay more attention to positive entries.

As there is not much new research done in the area of voxel based classification we chose a single Voxel Residual Network as a representative of this category. It still achieves accuracy comparable to the latest networks and has publicly available code. It is implemented in Theano with Lasagne and offers several models to train and make ensemble from. We opted for only one of these models as it is very time demanding to train even a single network. The model chosen is the model reported by authors as the best one and described in detail in the original paper. This approach still represents the state of the art in voxel based classification as ensemble of similar models reports 95.54% accuracy on ModelNet40 dataset which remains one of the highest reported.

3.1.3 Octree and Adaptive Octree Networks

Octree-based Convolutional Neural Network Wang et al. [2017] uses another data structure for representing 3D data – an octree Meagher [1980]. Octree is a tree where each node has exactly eight children so partitioning the space into finer and finer cubes. This basically means voxelization of the 3D model, but in this case only voxels on the borders are considered. This can be implemented efficiently and represented in a format suitable for GPU computation. In each leaf node a normal vector of the surface is stored. Authors then present an efficient way of performing convolutions on octrees and construct a hierarchical structure of shared layers for individual levels of the tree. The computation proceeds from the finest leaf octants and continues upwards to the root of the tree. This approach

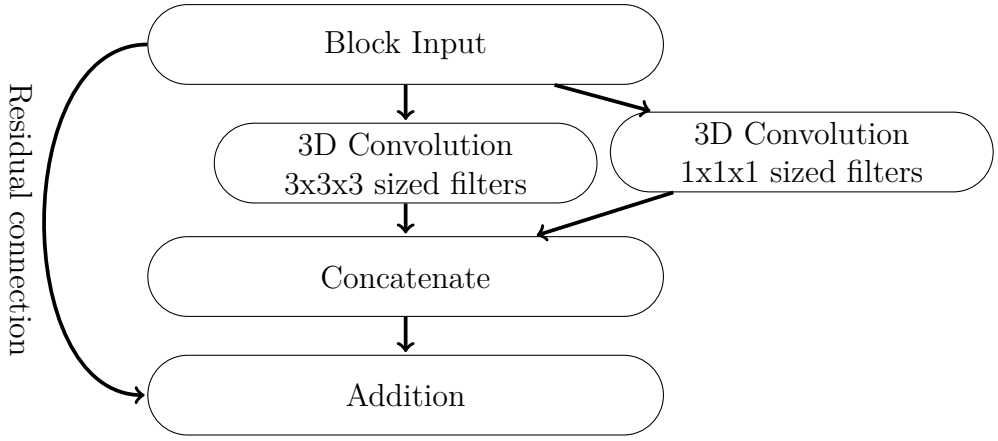


Figure 3.2: Example of a simple residual block

gives good results but the octrees are of a fixed maximum depth and therefore can waste memory on flat regions where a simple planar approximation would be sufficient. This problem is solved by Adaptive Octree Wang et al. [2018] representation which uses such planar patches as a representation in leaf nodes. Therefore flat areas of the original mesh can be represented by a simple leaf on a higher level of a tree, while more complex areas are subdivided into finer details. Authors offer an implementation of both networks in Caffe as well as tools for converting mesh data to octrees and adaptive octrees.

3.2 Multi-view Based Neural Networks

Another approach, harnessing the power of 2D convolutions and huge image datasets, are the so called Multi-view neural networks. A general setup of multi-view networks is as follows. They use rendered images of the 3D model from different angles as an input. These views are then passed to some pretrained image processing network and then some technique of combining features from different views is employed. Such techniques range from simple pooling across the views to employing recurrent neural network to process them as a sequence.

Multi-view approaches can be considered state of the art in this area as they achieve excellent results. For a fair comparison of these methods we use the same sets of images and twelve views of each 3D model rotated along the vertical axis.

3.2.1 Multi-view Convolutional Networks

For training a multi-view based network we need to fine tune some already pre-trained image recognition network. We use two different networks: smaller and older AlexNet Krizhevsky et al. [2012] and the state of the art deep network VGG Simonyan and Zisserman [2014]. This offers a simple method of 3D classification; we train the image network on the views of a rotated 3D model without any regard for the multi-view nature of the dataset. During evaluation we perform voting across the views. We chose VGG for this task as it performs better on image recognition tasks. We use a publicly available implementation of VGG (by machrisaa) with 19 weighted layers in Tensorflow. As we shall see later, even this simple approach yields results comparable with the most sophisticated networks.

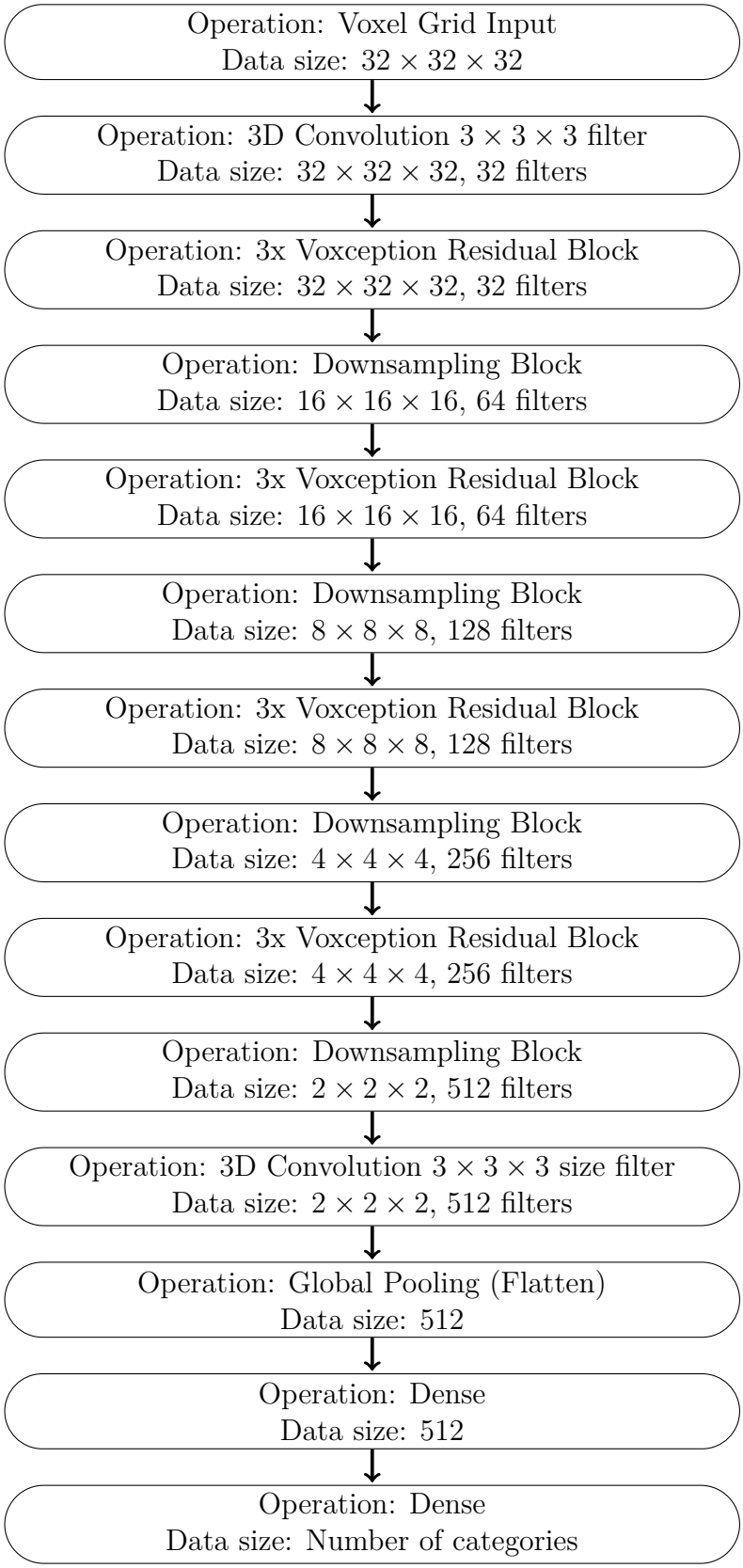


Figure 3.3: VRN architecture

The first multi-view approach to appear Su et al. [2015] uses shared convolutional layers to process individual views, then uses max pooling across the views to combine the features. Resulting features are fed to another convolutional network and then classified. We chose to test this approach as it is the first multi-view approach to achieve good results and it is simple enough to serve as baseline for similar approaches. From several available implementations of this network we have chosen a Tensorflow implementation. In this case, we use AlexNet as the pretrained image network, which is recommended by authors of the code and supported in the code structure.

The revisited but similar approach is used by Su et al. [2018], which divides the training phase in two stages. In the first stage the network is trained only using one view and later during the second phase pooling across the views is employed. Authors also explore different pre-trained image network architectures and different image rendering techniques improving accuracy of this method significantly. It uses a pre-trained VGG-11 convolutional network as its base. It offers a publicly available implementation in Pytorch which we have chosen to test as it promises some of the best accuracies achieved on ModelNet40 so far.

3.2.2 RotationNet

RotationNet Kanezaki et al. [2016] reports the highest achieved accuracy on the ModelNet40 dataset so it is of particular interest to us. It combines the multi-view classification with an unsupervised pose estimation task. Unlike other multi-view networks we do not provide information about the position of the viewpoints, i.e. they can be rotated arbitrarily hence the name of the network. To achieve this, a new category is added to the set of original classification categories. The meaning of this category is "this is not the correct viewpoint". All the possible rotations of the viewpoints are tried and the most probable according to predicted categories is chosen. Authors offer two implementations, one in Caffe, and another one in PyTorch. We have chosen to test this network as it reports very high accuracy on ModelNet40 and inherently contains pose estimation, which can be useful for future work.

3.2.3 Sequential Views to Sequential Labels

Sequential Views to Sequential Labels network Zhizhong et al. [2018] employs recurrent neural networks and treats multiple views as a sequence of images. It uses classic encoder-decoder architecture. In order to do this it treats its output not as a single vector but as a sequence of labels. It uses pretrained convolutional network which is fine-tuned on single-image classification task. We opted for the VGG-19 network already described above. The last fully connected layer of size 4096 is used as a feature vector for single views and fed into the encoder as a sequence. Both the encoder and decoder consists of GRU (subsection 2.1.3) cells and attention is used. Implementation in Tensorflow is available and we used it to test this network.

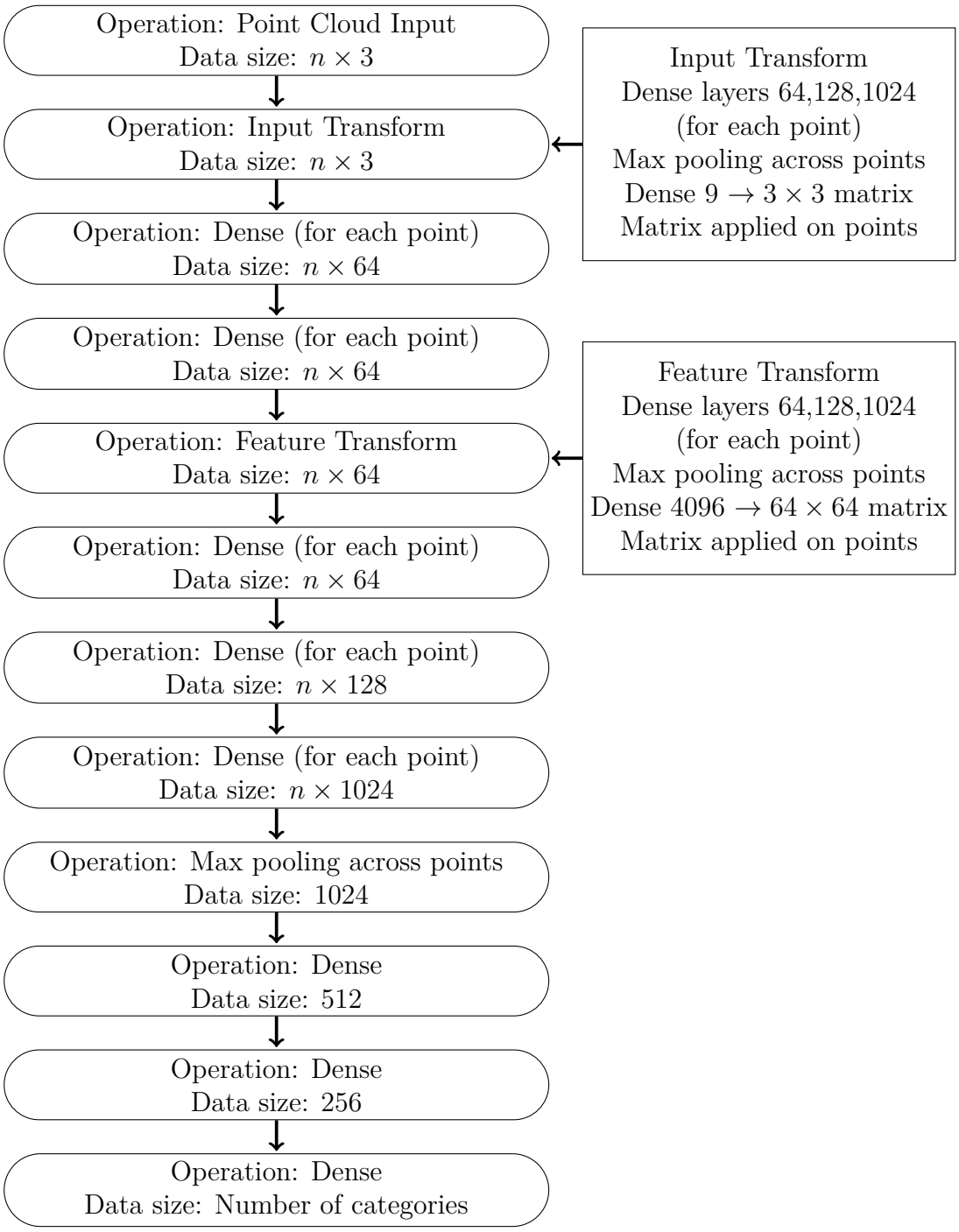


Figure 3.4: Multi-view architecture

3.3 Point Cloud Based Neural Networks

An altogether different representation of spatial data is a point cloud. A point cloud is an unordered set of points in the Euclidean space. This is a natural output format of laser scanning devices used by robots or autonomous cars and also in medical scanning. We can easily construct a point cloud from a mesh by sampling its faces. Point clouds are neither structured nor ordered as voxels or images are, which poses a problem to neural networks.

3.3.1 PointNet and PointNet++

The first network which successfully overcame all the difficulties of processing unordered point clouds was PointNet Qi et al. [2016]. Its main idea lies in using only symmetric functions i.e., functions for which the order of arguments does not matter. So each point is processed independently by a series of multi-layer perceptrons sharing weights. Then a global feature vector is constructed using max pooling, which is a symmetric function. Another important feature of PointNet are learnable geometric transformations which ensure some invariability to rotation or jittering of input point cloud. Rotation and jittering are also used as data augmentation during training.

Although PointNet achieves reasonable results it does not provide any mechanism for learning local features. Figure 3.5 PointNet++ Qi et al. [2017] presents a hierarchical structure inspired by convolutional neural networks which solves this very problem. It clusters close sets of points together and runs original PointNet on such neighborhoods. For this purpose iterative farthest point sampling and multi-resolution grouping (which ensures good representation for differently dense areas) are used. Thusly obtained local features are represented by the centroid of the original neighborhood and clustered again in a hierarchical manner. Finally a classic fully connected layer is employed for extracting global features and classification. There is an implementation for both networks available in Tensorflow and we tested both of them as they achieve reasonable accuracy and promise better scalability and are considerably faster than other methods.

3.3.2 Self-Organizing Network

The PointNet++ architecture lacks the ability to reveal the spatial distribution of the input point cloud during the hierarchical feature extraction. Self-Organizing Network Li et al. [2018] solves this problem by constructing *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 was shown to be a key feature. 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. There is an implementation in Pytorch available which contains also code for training self organizing maps from point clouds. We have chosen to test this network as it seems to offer significant improvement for a cost of only quick data preprocessing.

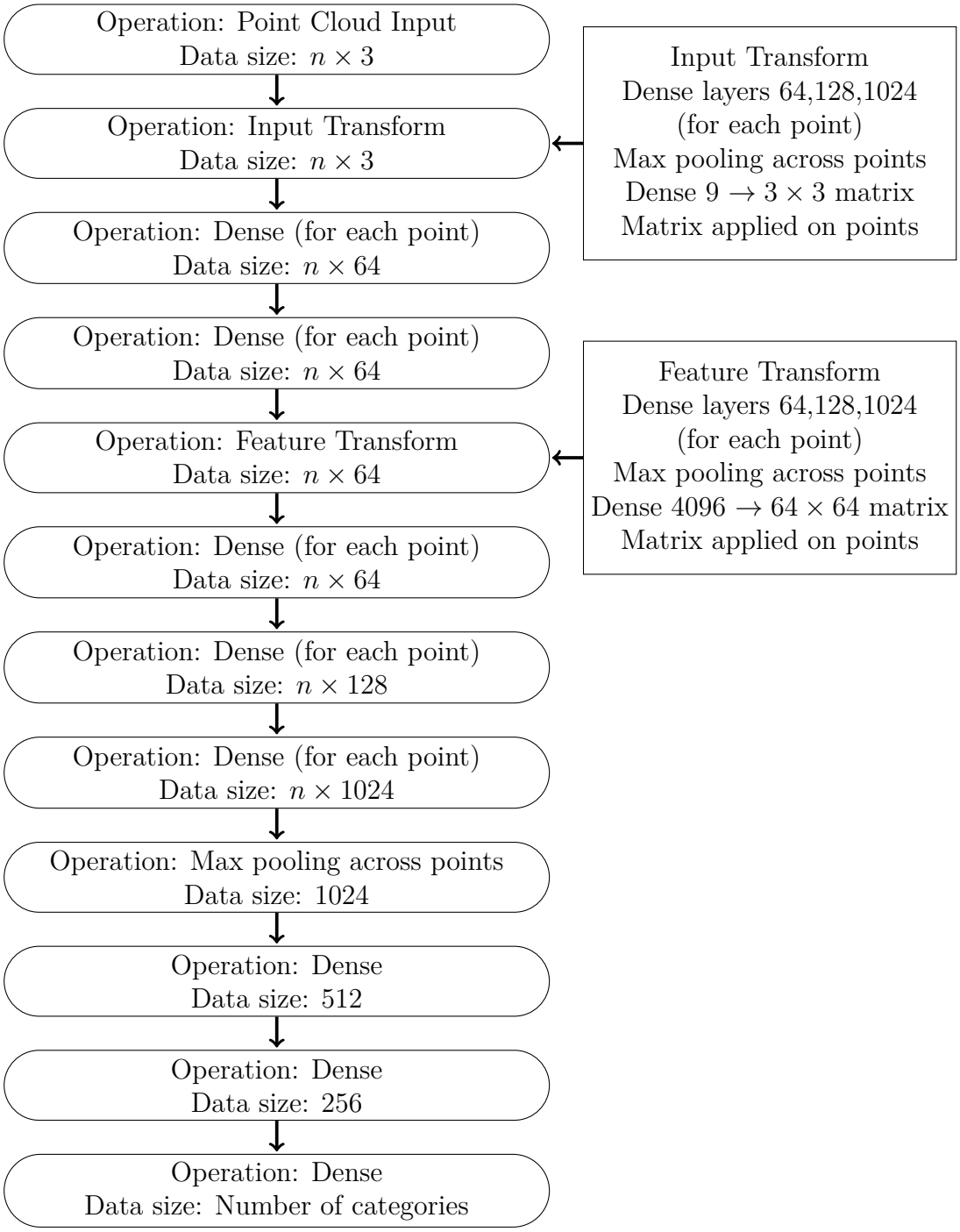


Figure 3.5: PointNet architecture. Layers labeled "for each point" are applied for each point separatly with same weights. n is number of input points.

3.3.3 KD Network

A kd-tree Bentley [1975] 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-net Klokov and Lempitsky [2017]. Firstly kd-tree is constructed over a point cloud. Then the tree is fed to a series of fully connected layers in a recursive manner starting in the leaf nodes and continuing to the root, where the global feature vector is extracted and used for classification. Weights of the fully connected layers are shared for nodes on the same level at the tree, where the tree is split along the same coordinate.

During training it uses several geometrical perturbations as 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. An implementation in Theano is supplied by the authors which also provides a framework for kd-tree construction from a point cloud.

3.3.4 Graph Based Convolutional Network

For the completeness' sake we mention a graph based approach Dominguez et al. [2018], which constructs a graph from a point cloud. Vertices of the graph are the original points and edges are constructed to the six nearest neighbors and sorted by their direction by some arbitrary sorting. Then special graph convolutions are applied repeatedly simplifying the structure of the graph and extracting local features. This approach is interesting from the theoretical standpoint but the training is very slow and it does not achieve state of the art results.

Category	Train	Test	Category	Train	Test
airplane	3235	810	jar	466	114
ashcan	275	68	knife	339	85
bag	66	17	lamp	1853	464
basket	82	21	laptop	360	91
bathtub	684	172	loudspeaker	1274	319
bed	184	49	mailbox	74	19
bench	1451	362	microphone	53	14
birdhouse	58	15	microwave	122	30
bookshelf	362	90	motorcycle	269	68
bottle	395	102	mug	171	43
bowl	146	39	piano	191	48
bus	751	188	pillow	76	20
cabinet	1247	315	pistol	247	60
camera	90	23	pot	439	110
can	84	21	printer	132	33
cap	44	12	remote control	52	14
car	2811	703	rifle	1864	467
cellular telephone	665	166	rocket	68	17
chair	5391	1354	skateboard	121	31
clock	521	130	sofa	2406	603
computer keyboard	51	13	stove	174	44
dishwasher	74	19	table	6702	1676
display	874	218	telephone	206	52
earphone	58	15	tower	98	25
faucet	593	149	train	311	78
file	230	59	vessel	1550	388
guitar	637	160	washer	133	34
helmet	129	33	Total	40939	10270

Table 3.2: List of ShapeNetCore categories and their counts.

Category	Train	Test	Category	Train	Test
airplane	626	100	mantel	284	100
bathtub	106	50	monitor	465	100
bed	515	100	night stand	200	86
bench	173	20	person	88	20
bookshelf	572	100	piano	231	100
bottle	335	100	plant	240	100
bowl	64	20	radio	104	20
car	197	100	range hood	115	100
chair	889	100	sink	128	20
cone	167	20	sofa	680	100
cup	79	20	stairs	124	20
curtain	138	20	stool	90	20
desk	200	86	table	392	100
door	109	20	tent	163	20
dresser	200	86	toilet	344	100
flower pot	149	20	tv stand	267	100
glass box	171	100	vase	475	100
guitar	155	100	wardrobe	87	20
keyboard	145	20	xbox	103	20
lamp	124	20	**Total**	**9843**	**2468**
laptop	149	20			

Table 3.3: List of ModelNet40 categories and their counts.

4. Methods

We begin this chapter by introducing the data used for our experiments as well as some additional sources for further research. We continue by briefly discussing the methods of converting 3D meshes to various other representations and we end with discussing our software decisions.

4.1 Datasets

In this section we introduce datasets of 3D models which we used or considered to use for testing.

4.1.1 ModelNet40

ModelNet Wu et al. [2014] is one of the most well-known and commonly used dataset containing annotated 3D models in a mesh format. It was developed by a team at Princeton University. Its subset, called ModelNet40, in particular is used as a baseline for testing different approaches. We therefore decided to use this dataset as a starting point for our evaluations. ModelNet40 contains forty different categories and 12 311 individual models. The dataset has an official split to training and testing subsets, which we adhere to in all cases. The test set contains 2648 models and is never used for training.

The models in original ModelNet40 are not aligned and have widely different scales. Therefore when preprocessing the data for neural networks we rescale all models to fit a unit sphere and we use manually aligned version of the dataset Sedaghat et al. [2016]. Also the categories are not equally populated for example there are over 700 airplane models and only over 100 wardrobe models. The exact numbers of models in particular categories can be found in Table 4.1. ModelNet40 contains files in .off format so our scripts have to be able to read this particular format. The dataset is available to download for academic purposes.

4.1.2 ShapeNetCore

ShapeNet Chang et al. [2015] is an ongoing effort to establish a richly-annotated, large-scale dataset of 3D shapes. ShapeNet is a collaborative effort between researchers at Princeton, Stanford and Toyota Technological Institute at Chicago. We used its subset called ShapeNetCore which contains 51 209 individual models in 55 categories. There is also an official split to training, test and validation sets. However, this split does not contain all models and is not divided uniformly. We therefore decided to construct our own split – 80% of models in each category is assigned to the training set and the rest to the test set. By doing this we obtained a training set with 40 939 models and a test set with 10 270 models.

Table 4.2 lists the exact numbers of models in particular categories. During our exploration of the dataset we noticed that some models are assigned to more than one category so we were forced to choose one of them somewhat arbitrarily. You can find our final split into sets and categories in csv file [TODO:priloha x].

Category	Train	Test	Category	Train	Test
airplane	626	100	mantel	284	100
bathtub	106	50	monitor	465	100
bed	515	100	night stand	200	86
bench	173	20	person	88	20
bookshelf	572	100	piano	231	100
bottle	335	100	plant	240	100
bowl	64	20	radio	104	20
car	197	100	range hood	115	100
chair	889	100	sink	128	20
cone	167	20	sofa	680	100
cup	79	20	stairs	124	20
curtain	138	20	stool	90	20
desk	200	86	table	392	100
door	109	20	tent	163	20
dresser	200	86	toilet	344	100
flower pot	149	20	tv stand	267	100
glass box	171	100	vase	475	100
guitar	155	100	wardrobe	87	20
keyboard	145	20	xbox	103	20
lamp	124	20	**Total**	**9843**	**2468**
laptop	149	20			

Table 4.1: List of ModelNet40 categories and their counts.

All models in ShapeNetCore are already aligned and rescaled to fit a unit sphere. The categories are not distributed equally at all as you can see from the table. The dataset is also freely available to download for academic purposes.

4.1.3 Other 3D Datasets

In this section we mention several publicly available datasets containing 3D models which can be used for further research.

Both ModelNet and ShapeNet contain much more models than the standardized subsets we used for our evaluation. Therefore there is an option to download the whole datasets or to construct custom subsets.

A Large Dataset of Object Scans Choi et al. [2016] is a dataset focusing on video scan to 3D model reconstruction but we suppose it can be used for learning classification as well.

ObjectNet3D Xiang et al. [2016] focuses on image to 3D model reconstruction and contains a large number of 3D models that can be used for classification training.

SUNCG dataset Song et al. [2017] contains entire indoor scenes but is annotated on the level of single objects and therefore can be parsed and used for classification.

SceneNN Hua et al. [2016] dataset contains a large number of scenes which are richly annotated and can be split into single objects similarly to the previous dataset.

Category	Train	Test	Category	Train	Test
airplane	3235	810	jar	466	114
ashcan	275	68	knife	339	85
bag	66	17	lamp	1853	464
basket	82	21	laptop	360	91
bathtub	684	172	loudspeaker	1274	319
bed	184	49	mailbox	74	19
bench	1451	362	microphone	53	14
birdhouse	58	15	microwave	122	30
bookshelf	362	90	motorcycle	269	68
bottle	395	102	mug	171	43
bowl	146	39	piano	191	48
bus	751	188	pillow	76	20
cabinet	1247	315	pistol	247	60
camera	90	23	pot	439	110
can	84	21	printer	132	33
cap	44	12	remote control	52	14
car	2811	703	rifle	1864	467
cellular telephone	665	166	rocket	68	17
chair	5391	1354	skateboard	121	31
clock	521	130	sofa	2406	603
computer keyboard	51	13	stove	174	44
dishwasher	74	19	table	6702	1676
display	874	218	telephone	206	52
earphone	58	15	tower	98	25
faucet	593	149	train	311	78
file	230	59	vessel	1550	388
guitar	637	160	washer	133	34
helmet	129	33	Total	40939	10270

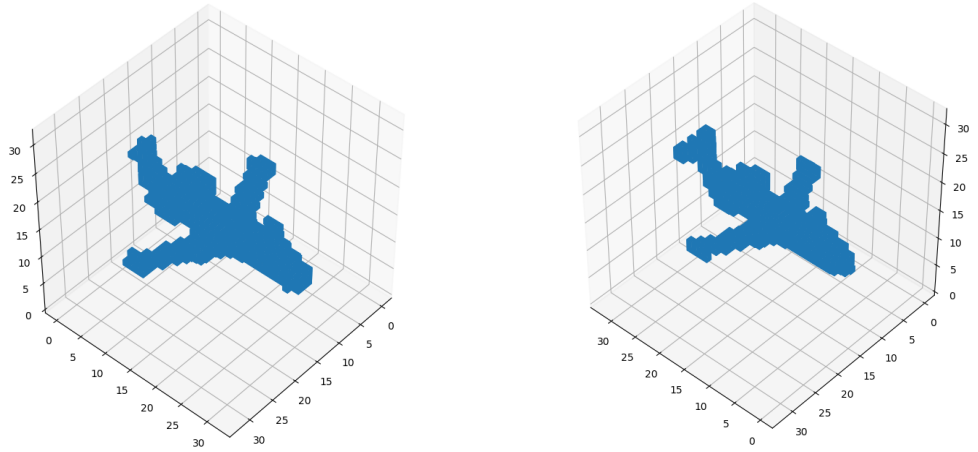
Table 4.2: List of ShapeNetCore categories and their counts.

4.2 Data Conversion

As mentioned in previous chapters, mesh files, in which most existing 3D models are saved, are not suitable for direct processing by neural networks. Therefore we have to be able to convert meshes to voxels, images or point clouds.

4.2.1 Mesh to Voxels

In order to use voxel based systems we need to convert mesh files to voxel occupancy grids. For this purpose we have chosen the OpenVDB library, which is free, open-source and offers python scripting Museth et al. [2013]. OpenVDB offers voxelization as one of its core functions, implemented in C++. We supply python scripts for voxelization of ModelNet40 and ShapeNetCore datasets using python multiprocessing to parallelize the computation. Still it can take several hours to process the whole dataset as we need to voxelize multiple rotations for each model.



(a) Original voxel representation provided by authors of VRN

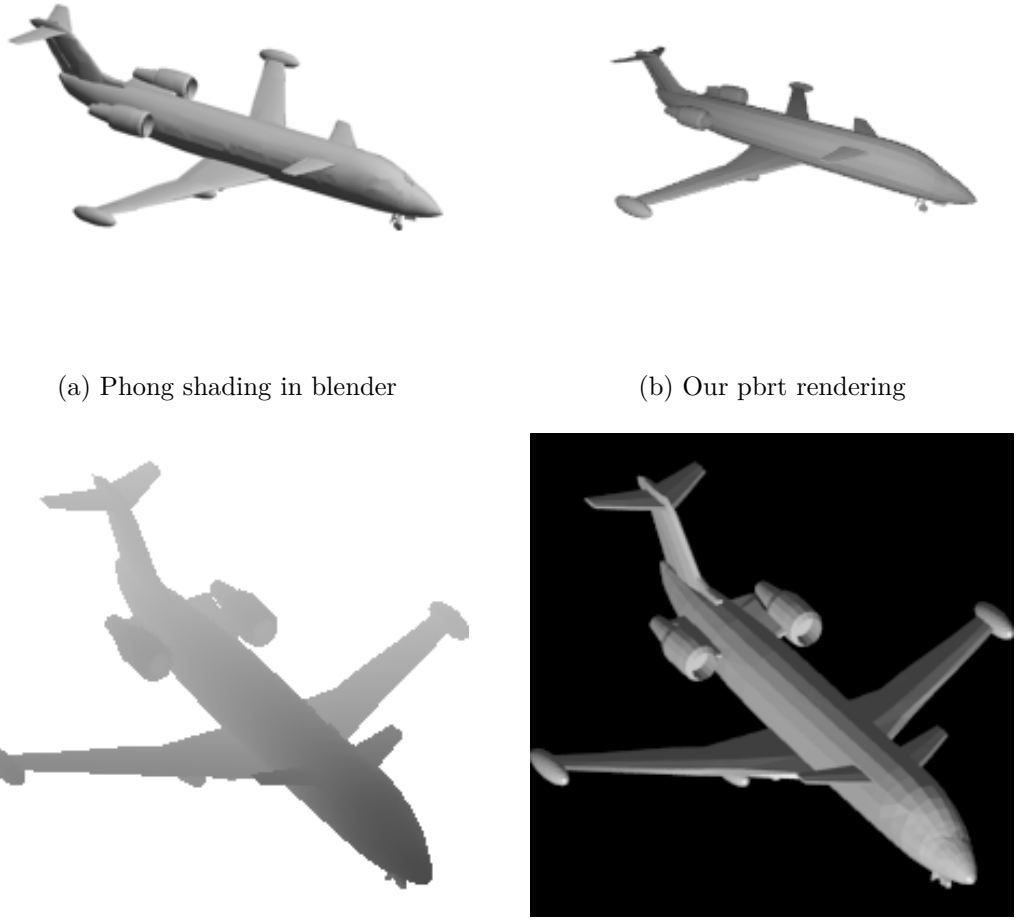
(b) Our voxelization using OpenVDB

Figure 4.1: Illustration of voxel representation

4.2.2 Mesh to Images

For multi-view based neural network we have to be able to render images taken from arbitrary viewpoints of a 3D mesh. Firstly we tried to replicate results used by the authors of MVCNN Su et al. [2015] and we used pbrt Pharr and Humphreys [2010], which is physically based rendering software with publicly available code. This turned out to be a plausible approach. We also used the original blender scripts using phong shading Bishop and Weimer [1986] in blender to render the images. Later in our research we found blender scripts from the authors of MVCNN2 Su et al. [2018]. They provide two different rendering options – shaded images and depth images. These work faster and achieve better accuracy than both ours and phong shaded images. In our framework we provide

both approaches implemented with python scripts and multiprocessing support. Figure 4.2 shows one of the airplane models rendered by the four different scripts.



(a) Phong shading in blender

(b) Our pbrt rendering

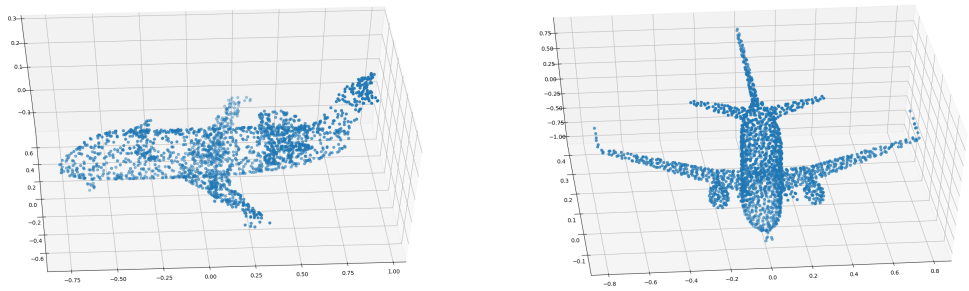
(c) Depth image in blender

(d) Shaded image in blender

Figure 4.2: Illustration of different image representations

4.2.3 Mesh to Point Cloud

For the use of point cloud based neural networks we have to be able to construct a point cloud from a 3D mesh. This is a much more straightforward problem than the conversions described above. A point cloud is created by random sampling from the polygons forming the mesh. First, we select a polygon with a probability proportional to the area of that polygon. Then we sample a random point within the selected polygon by generating random barycentric coordinates. We provide a python script with support for multiprocessing and this is sufficiently fast for our purposes.



(a) Original point cloud representation provided by the authors of PointNet

(b) Our point cloud representation

Figure 4.3: Illustration of point cloud representation

4.3 Technical Setup

This section provides a brief summary of software choices we made. For information about prerequisites and structure of our framework, please consult the manual [TODO příloha].

As one of our main goals is to provide the academical community with easily runnable code, we opted for a solution using Docker Merkel [2014]. Docker is a program used to run software packages called containers. Containers are isolated bundles of software, libraries and configurations. The specification of a container is called an image. An image contains a so called Dockerfile which defines the image, allowing automatic installation of all dependencies, setting up configurations, etc. Every neural network and data conversion package is thus a completely independent piece of software which can be run almost without any prerequisites. We consider this to be one of the main contributions of our work.

As all the machine learning frameworks we encountered support handling by python scripts and python is the most commonly used programming language in machine learning and artificial intelligence, we naturally use it for most of our code. We also preferred libraries for data conversion which support python. Some of the neural networks are implemented in such a way that they accept a purely pythonic file format as their input. So library not supporting python would require one more data conversion step.

We currently support only Linux, but Docker can be run on Windows as well and we believe that our framework can be extended to run on Windows without great difficulties.

5. Experiments and Results

Acronym	Reported	Reported avg	Measured	Measured avg
Voxel				
VoxNet	83 %	–	–	–
ORION	89.7 %	–	–	–
FusionNet	90.8 %	–	–	
VRN	91.33 %	–	88.65 %	86.72 %
O-CNN	90.6 %	–	83.47 %	78.46 %
AO-CNN	90.5 %	–	87.44 %	84.09 %
Multi-view				
VGG ()	–	–	89.95 %	87.73 %
MVCNN (shaded)	90.1 %	–	88.65 %	86.24 %
MVCNN2 (depth)	95.0 %	92.4 %	91.41 %	89.13 %
RotationNet(shaded)	97.37%	–	92.1 %	89.86 %
Seq2Seq (shaded)	93.31 %	–	90.84 %	88.54 %
Point cloud				
PointNet	89.2 %	86.2 %	84.0 %	78.81 %
PointNet++	91.9 %	–	88.49 %	85.3 %
SO-Net	93.4 %	87.3 %	89.14 %	85.5 %
KD-Net	91.8 %	88.5 %	88.57 %	83.84 %
GraphNet	91.13 %	–	–	–

Table 5.1: List of ModelNet40 accuracies.

Acronym	Measured	Measured avg
VGG (pbrt)	87.93 %	84.75 %
VGG (depth)	89.14 %	85.66 %
VGG (shaded)	89.95 %	87.73 %
MVCNN (pbrt)	87.8 %	85.54 %
MVCNN (depth)	87.6 %	85.07 %
MVCNN (shaded)	88.65 %	86.24 %
MVCNN2 (pbrt)	90.68 %	88.44 %
MVCNN2 (depth)	91.41 %	89.13 %
MVCNN2 (shaded)	90.84 %	88.38 %
RotationNet (pbrt)	91.09 %	87.78 %
RotationNet (depth)	91.57 %	88.62 %
RotationNet (shaded)	92.1 %	89.86 %
Seq2Seq (pbrt)	88.13 %	84.81 %
Seq2Seq (depth)	89.14 %	86.26 %
Seq2Seq (shaded)	90.84 %	88.54 %

Table 5.2: Comparison of multi-view methods.

Acronym	Epochs	Total hours	One epoch(minutes)	Examples per second
Voxel				
VRN	20	120	394	0.5
O-CNN	200	4	1.15	140
AO-CNN	200	2	0.65	250
Multi-view				
VGG	60	20	25	6
MVCNN	200	8	2.33	70
MVCNN2	60	14	13.33	12
RotationNet	200	8	2.73	60
Seq2Seq	300	1	0.25	650
Point cloud				
PointNet	200	10	2.22	70
PointNet++	200	4	1.07	150
SO-Net	400	6	0.9	180
KD-Net	200	8	2.61	60

Table 5.3: Table of approximate training times on ModelNet40.

Acronym	Size of the model(MB)
Voxel	
VRN	50
O-CNN	2
AO-CNN	2
Multi-view	
VGG	500
MVCNN	800
MVCNN2	500
RotationNet	230
Seq2Seq	30
Point cloud	
PointNet	40
PointNet++	18
SO-Net	10
KD-Net	8

Table 5.4: Approximate sizes of saved models, which roughly corresponds to number of trainable parameters of the model.

Category	Accuracy	Test cases	Category	Accuracy	Test cases
flower pot	14.60%	(20 cases)	door	90.40%	(20 cases)
wardrobe	63.00%	(20 cases)	piano	90.96%	(100 cases)
cup	68.00%	(20 cases)	range hood	91.00%	(100 cases)
night stand	70.14%	(86 cases)	tent	92.60%	(20 cases)
bench	73.20%	(20 cases)	bookshelf	93.04%	(100 cases)
radio	73.20%	(20 cases)	cone	93.60%	(20 cases)
xbox	75.40%	(20 cases)	glass box	93.92%	(100 cases)
stool	76.40%	(20 cases)	mantel	94.92%	(100 cases)
table	76.44%	(100 cases)	sofa	95.24%	(100 cases)
dresser	76.70%	(86 cases)	bottle	95.32%	(100 cases)
vase	77.08%	(100 cases)	person	96.20%	(20 cases)
desk	79.35%	(86 cases)	monitor	96.44%	(100 cases)
sink	80.00%	(20 cases)	chair	97.00%	(100 cases)
tv stand	82.04%	(100 cases)	bed	97.64%	(100 cases)
bathtub	83.68%	(50 cases)	car	98.56%	(100 cases)
plant	83.68%	(100 cases)	toilet	98.72%	(100 cases)
lamp	83.80%	(20 cases)	keyboard	99.00%	(20 cases)
stairs	87.00%	(20 cases)	guitar	99.48%	(100 cases)
bowl	88.40%	(20 cases)	laptop	99.60%	(20 cases)
curtain	88.80%	(20 cases)	airplane	99.72%	(100 cases)

Table 5.5: Averege accuracies per category of ModelNet40, sorted from worst to best.

6. Conclusions

6.1 Summary

6.2 Further Work

Bibliography

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv:1409.0473 [cs, stat]*, September 2014. URL `http://arxiv.org/abs/1409.0473`. arXiv: 1409.0473.

Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, September 1975. ISSN 00010782. doi: 10.1145/361002.361007. URL `http://portal.acm.org/citation.cfm?doid=361002.361007`.

Bishop and Weimer. Fast Phong Shading. *Computer Graphics (20) 4 pp. 103-106*, 20, 1986. doi: 10.1145/15886.15897.

Andrew Brock, Theodore Lim, J. M. Ritchie, and Nick Weston. Generative and Discriminative Voxel Modeling with Convolutional Neural Networks. *arXiv:1608.04236 [cs, stat]*, August 2016. URL `http://arxiv.org/abs/1608.04236`. arXiv: 1608.04236.

Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3d Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *arXiv:1406.1078 [cs, stat]*, June 2014. URL `http://arxiv.org/abs/1406.1078`. arXiv: 1406.1078.

Sungjoon Choi, Qian-Yi Zhou, Stephen Miller, and Vladlen Koltun. A Large Dataset of Object Scans. *arXiv:1602.02481*, 2016. URL `http://redwood-data.org/3dscan/index.html`.

François Chollet and others. *Keras*. 2015. URL `https://keras.io`.

Sander Dieleman, Jan Schlüter, Colin Raffel, Eben Olson, Søren Kaae Sønderby, Daniel Nouri, Daniel Maturana, Martin Thoma, Eric Battenberg, Jack Kelly, Jeffrey De Fauw, Michael Heilman, Diogo Moitinho de Almeida, Brian McFee, Hendrik Weideman, Gábor Takács, Peter de Rivaz, Jon Crall, Gregory Sanders, Kashif Rasul, Cong Liu, Geoffrey French, and Jonas Degrave. *Lasagne: First release.* August 2015. doi: 10.5281/zenodo.27878. URL `http://dx.doi.org/10.5281/zenodo.27878`.

Miguel Dominguez, Rohan Dhamdhere, Atir Petkar, Saloni Jain, Shagan Sah, and Raymond Ptucha. *General-Purpose Deep Point Cloud Feature Extractor*. March 2018. doi: 10.1109/WACV.2018.00218.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL `https://www.deeplearningbook.org/`.

Vishakh Hegde and Reza Zadeh. FusionNet: 3d Object Classification Using Multiple Data Representations. *arXiv:1607.05695 [cs]*, July 2016. URL http://arxiv.org/abs/1607.05695. arXiv: 1607.05695.

Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural Networks for Machine Learning: Lecture 6a Overview of mini-batch gradient descent. URL http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

Sepp Hochreiter and Jürgen Schmidhuber. Long Short-term Memory. *Neural computation*, 9:1735–80, 1997. doi: 10.1162/neco.1997.9.8.1735.

Binh-Son Hua, Quang-Hieu Pham, Duc Thanh Nguyen, Minh-Khoi Tran, Lap-Fai Yu, and Sai-Kit Yeung. SceneNN: A Scene Meshes Dataset with aNNotations. In *International Conference on 3D Vision (3DV)*, 2016. URL http://www.scenenn.net/.

Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Weinberger. Deep Networks with Stochastic Depth. *arXiv:1603.09382 [cs]*, March 2016. URL http://arxiv.org/abs/1603.09382. arXiv: 1603.09382.

Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv:1502.03167 [cs]*, February 2015. URL http://arxiv.org/abs/1502.03167. arXiv: 1502.03167.

Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional Architecture for Fast Feature Embedding. In *Proceedings of the 22Nd ACM International Conference on Multimedia*, MM '14, pages 675–678, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-3063-3. doi: 10.1145/2647868.2654889. URL http://doi.acm.org/10.1145/2647868.2654889. event-place: Orlando, Florida, USA.

Asako Kanezaki, Yasuyuki Matsushita, and Yoshifumi Nishida. RotationNet: Joint Object Categorization and Pose Estimation Using Multiviews from Unsupervised Viewpoints. *arXiv:1603.06208 [cs]*, March 2016. URL http://arxiv.org/abs/1603.06208. arXiv: 1603.06208.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*, December 2014. URL http://arxiv.org/abs/1412.6980. arXiv: 1412.6980.

Roman Klokov and Victor Lempitsky. Escape from Cells: Deep Kd-Networks for the Recognition of 3d Point Cloud Models. *arXiv:1704.01222 [cs]*, April 2017. URL http://arxiv.org/abs/1704.01222. arXiv: 1704.01222.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA, 2012. Curran Associates Inc. URL http://dl.acm.org/citation.cfm?id=2999134.2999257. event-place: Lake Tahoe, Nevada.

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4):541–551, 1989.

Jiaxin Li, Ben M. Chen, and Gim Hee Lee. SO-Net: Self-Organizing Network for Point Cloud Analysis. *arXiv:1803.04249 [cs]*, March 2018. URL `http://arxiv.org/abs/1803.04249`. arXiv: 1803.04249.

machrisaa. tensorflow-vgg. URL `https://github.com/machrisaa/tensorflow-vgg`.

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. URL `http://tensorflow.org/`.

D. Maturana and S. Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *IROS 2015*, 2015.

Donald Meagher. *Octree Encoding: A New Technique for the Representation, Manipulation and Display of Arbitrary 3-D Objects by Computer*. October 1980.

Dirk Merkel. Docker: Lightweight Linux Containers for Consistent Development and Deployment. *Linux J.*, 2014(239), March 2014. ISSN 1075-3583. URL `http://dl.acm.org/citation.cfm?id=2600239.2600241`.

Ken Museth, Jeff Lait, John Johanson, Jeff Budsberg, Ron Henderson, Mihai Alden, Peter Cucka, David Hill, and Andrew Pearce. OpenVDB: An Open-source Data Structure and Toolkit for High-resolution Volumes. In *ACM SIGGRAPH 2013 Courses*, SIGGRAPH '13, pages 19:1–19:1, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2339-0. doi: 10.1145/2504435.2504454. URL `http://doi.acm.org/10.1145/2504435.2504454`. event-place: Anaheim, California.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS-W*, 2017.

Matt Pharr and Greg Humphreys. *Physically Based Rendering, Second Edition: From Theory To Implementation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edition, 2010. ISBN 0-12-375079-2 978-0-12-375079-2.

Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. PointNet: Deep Learning on Point Sets for 3d Classification and Segmentation.

arXiv:1612.00593 [cs], December 2016. URL http://arxiv.org/abs/1612.00593. arXiv: 1612.00593.

Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. *arXiv:1706.02413 [cs]*, June 2017. URL http://arxiv.org/abs/1706.02413. arXiv: 1706.02413.

Frank Rosenblatt. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan Books, Washington DC, 1961.

Nima Sedaghat, Mohammadreza Zolfaghari, Ehsan Amiri, and Thomas Brox. Orientation-boosted Voxel Nets for 3d Object Recognition. *arXiv:1604.03351 [cs]*, April 2016. URL http://arxiv.org/abs/1604.03351. arXiv: 1604.03351.

Frank Seide and Amit Agarwal. CNTK: Microsoft's Open-Source Deep-Learning Toolkit. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 2135–2135, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4232-2. doi: 10.1145/2939672.2945397. URL http://doi.acm.org/10.1145/2939672.2945397. event-place: San Francisco, California, USA.

K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR*, abs/1409.1556, 2014.

Shuran Song, Fisher Yu, Andy Zeng, Angel X Chang, Manolis Savva, and Thomas Funkhouser. Semantic Scene Completion from a Single Depth Image. *Proceedings of 29th IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. Multi-view Convolutional Neural Networks for 3d Shape Recognition. *arXiv:1505.00880 [cs]*, May 2015. URL http://arxiv.org/abs/1505.00880. arXiv: 1505.00880.

Jong-Chyi Su, Matheus Gadelha, Rui Wang, and Subhransu Maji. A Deeper Look at 3d Shape Classifiers. *arXiv:1809.02560 [cs]*, September 2018. URL http://arxiv.org/abs/1809.02560. arXiv: 1809.02560.

I Sutskever, J Martens, G Dahl, and G Hinton. On the importance of initialization and momentum in deep learning. *30th International Conference on Machine Learning, ICML 2013*, pages 1139–1147, 2013.

Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alex Alemi. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. *arXiv:1602.07261 [cs]*, February 2016. URL http://arxiv.org/abs/1602.07261. arXiv: 1602.07261.

Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016. URL `http://arxiv.org/abs/1605.02688`.

Peng-Shuai Wang, Yang Liu, Yu-Xiao Guo, Chun-Yu Sun, and Xin Tong. O-CNN: Octree-based Convolutional Neural Networks for 3d Shape Analysis. *ACM Transactions on Graphics (SIGGRAPH)*, 36(4), 2017.

Peng-Shuai Wang, Yang Liu, Yu-Xiao Guo, Chun-Yu Sun, and Xin Tong. Adaptive O-CNN: A Patch-based Deep Representation of 3d Shapes. *ACM Transactions on Graphics (SIGGRAPH Asia)*, 37(6), 2018.

Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d ShapeNets: A Deep Representation for Volumetric Shapes. *arXiv:1406.5670 [cs]*, June 2014. URL `http://arxiv.org/abs/1406.5670`. arXiv: 1406.5670.

Yu Xiang, Wonhui Kim, Wei Chen, Jingwei Ji, Christopher Choy, Hao Su, Roozbeh Mottaghi, Leonidas Guibas, and Silvio Savarese. ObjectNet3d: A Large Scale Database for 3d Object Recognition. In *European Conference Computer Vision (ECCV)*, 2016. URL `http://cvgl.stanford.edu/projects/objectnet3d/`.

Han Zhizhong, Shang Mingyang, and Liu Zhenbao. SeqViews2seqlabels: Learning 3d Global Features via Aggregating Sequential Views by RNN With Attention. *IEEE Transactions on Image Processing*, 28(2):658 – 672, September 2018.

List of Figures

List of Tables

List of Abbreviations

A. Attachments

A.1 First Attachment