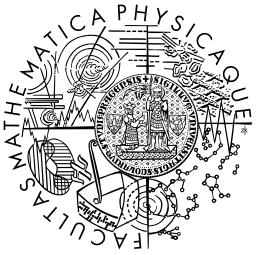

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Bc. Miroslav Krabec

Survey of deep neural networks for classification of 3D models

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 Inteligence

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: Survey of deep neural networks for classification of 3D models

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

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 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 and their relative performance has never been objectively evaluated.

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

1.2 Problem Statement, Scope of the Thesis

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

We can define the problem formally as follows: we are given 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 and y_i is a numerical representation of corresponding label. Each shape belongs to exactly one class. The goal of classification can be formulated as learning a parametric model $P : X \rightarrow Y$ 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.

We limit the scope of this thesis in the following ways. We will perform classification only on aligned 3D shapes, as no 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.

1.3 Thesis Outline

In this chapter 1 we presented basics of the problem as well as our motivation for this work. In the second chapter 2 we provide a brief introduction to artificial neural networks. In the third chapter 3 we introduce different approaches to 3D classification and their implementations. In the fourth chapter 4 we discuss chosen methods, describe datasets, procedures of data conversion and technical setup of our framework. In chapter five 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

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 principles of artificial neural networks. For more in depth information we recommend Goodfellow et al. [2016]. 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.

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.

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:

$$output_j = f(\sum_{i \in inputs} w_{ij} * i + b_j)$$

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 . 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 $ReLU(x) = max(0, x)$.

We can insert one or more layers of neurons between input and output obtaining a *Multi-layered perceptron*. Neuron from $i-th$ layer gets its input from all neurons from $(i-1)-th$ layer and passes its output to all neurons in $(i+1)-th$ layer. Therefore this type of layer is called *fully connected layer* or sometimes dense layer. We can also rewrite all the weights to a matrix form and obtain the so called weight matrix. Output of a fully connected layer can be expressed as follows:

$$output_i = f(W_i * output_{i-1} + b_i)$$

Where f is an activation function, W_i is the weight matrix and b_i is a vector of biases.

Multi layered perceptrons can be trained for variety of tasks using a gradient descent algorithm by backpropagating the error of the trained task through the network in direction from outputs to inputs. 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.

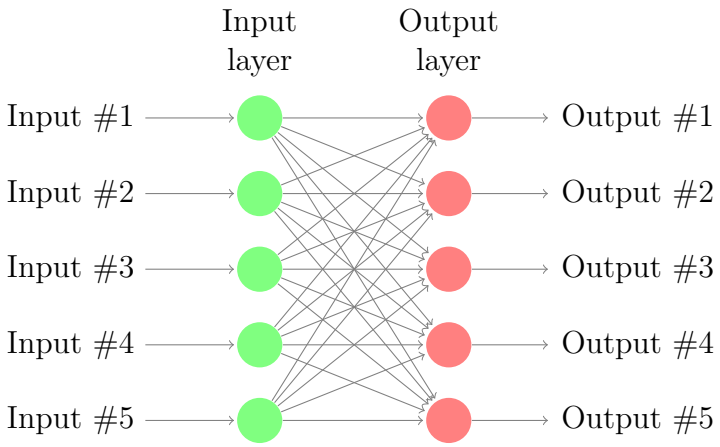

Figure 2.1: Single-layer perceptron

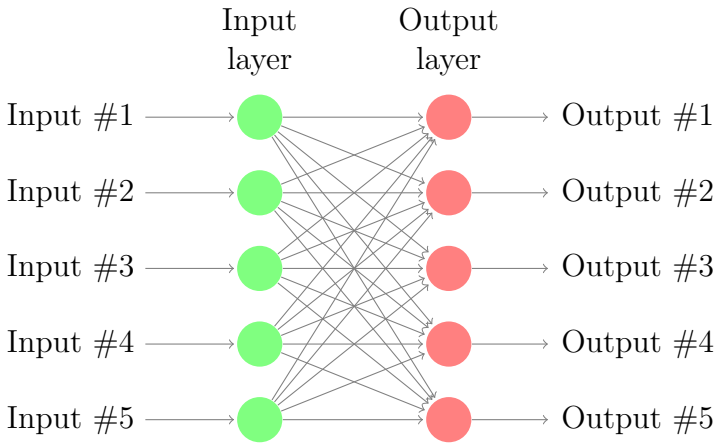

Figure 2.2: Single-layer perceptron

2.1.2 Convolutional Neural Networks

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

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 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:

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

Where x and y are coordinates of a pixel in the picture, d_h and d_w are sizes of the sliding window.

Convolutional layers are much more efficient than fully connected layers as they share their parameters across the image and have been very successful in variety of image, video and natural language processing tasks.

2.1.3 Recurrent Neural Networks

Another widely used class of artificial neural networks are *recurrent neural networks*. 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 RNN is *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]. 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 networks we have tested uses recurrent neural networks 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 some 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. We will see several examples of data augmentation in later chapters.

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 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] has 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 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 to reduce the vanishing gradient problem.

2.2 Deep Learning Frameworks

In recent years several software frameworks for machine learning in general and for deep learning in particular are being 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 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 team at Google. It is a symbolic math library and implements all the standard neural layers and also 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 writing any code just network definition. It does not seem to be developing as quickly as above mentioned frameworks. There was also Caffe2 developed by Facebook, but it was merged into PyTorch.

Theano Al-Rfou et al. [2016] is a Python library for manipulating and evaluating mathematical expressions, primarily developed by a 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 a 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 of classifying 3D shapes by means of artificial neural networks were 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 neural network so we divide the networks according to format which they use as their input: voxel-based, multi-view, and point-cloud based. 3.1 shows a list of neural networks described in this chapter. There is a citation of an original paper, a framework of the publicly available code and if we included the network in our testing.

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.

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 n rotations of the input object to the network and then uses pooling across the rotation 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 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.

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 (Section XYZ), residual connections (Section XYZ) 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 3x3x3 convolution and the result of a 1x1x1 convolution, so the network can learn which of these filters to apply. 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 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 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 simple leaf on a higher level of a tree, while complicated areas are divided to 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 for training and testing 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 a single rendered view of a rotated 3D model and then during evaluation we perform voting across all views. We chose VGG for this task as it performs better on image recognition tasks. We use a publicly available implementation of VGG machrisaa with 19 weighted layers in Tensorflow. As we shall see later, even this simple approach yields results comparable with the most sophisticated networks.

The first and simplest multi-view approach 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. As a pretrained image network we use AlexNet 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 its 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 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 the second 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 images classification task. We opted for 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 encoder and decoder consists of GRU cells and attention is used. Implementation in Tensorflow is available and we used it to test this network.

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 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 nor structured neither 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. 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 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.

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 splitting 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 theoretical standpoint but the training is very slow and it does not achieve state of the art results.

4. Methods

5. Experiments and Results

6. Conclusions

6.1 Summary

6.2 Further Work

Bibliography

Rami Al-Rfou, Guillaume Alain, Amjad Almahairi, Christof Angermueller, Dzmitry Bahdanau, Nicolas Ballas, Frédéric Bastien, Justin Bayer, Anatoly Belikov, Alexander Belopolsky, Yoshua Bengio, Arnaud Bergeron, James Bergstra, Valentin Bisson, Josh Bleecher Snyder, Nicolas Bouchard, Nicolas Boulanger-Lewandowski, Xavier Bouthillier, Alexandre de Brébisson, Olivier Breuleux, Pierre-Luc Carrier, Kyunghyun Cho, Jan Chorowski, Paul Christiano, Tim Cooijmans, Marc-Alexandre Côté, Myriam Côté, Aaron Courville, Yann N. Dauphin, Olivier Delalleau, Julien Demouth, Guillaume Desjardins, Sander Dieleman, Laurent Dinh, Mélanie Ducoffe, Vincent Dumoulin, Samira Ebrahimi Kahou, Dumitru Erhan, Ziye Fan, Orhan Firat, Mathieu Germain, Xavier Glorot, Ian Goodfellow, Matt Graham, Caglar Gulcehre, Philippe Hamel, Iban Harlouchet, Jean-Philippe Heng, Balázs Hidasi, Sina Honari, Arjun Jain, Sébastien Jean, Kai Jia, Mikhail Korobov, Vivek Kulkarni, Alex Lamb, Pascal Lamblin, Eric Larsen, César Laurent, Sean Lee, Simon Lefrancois, Simon Lemieux, Nicholas Léonard, Zhouhan Lin, Jesse A. Livezey, Cory Lorenz, Jeremiah Lowin, Qianli Ma, Pierre-Antoine Manzagol, Olivier Mastropietro, Robert T. McGibbon, Roland Memisevic, Bart van Merriënboer, Vincent Michalski, Mehdi Mirza, Alberto Orlandi, Christopher Pal, Razvan Pascanu, Mohammad Pezeshki, Colin Raffel, Daniel Renshaw, Matthew Rocklin, Adriana Romero, Markus Roth, Peter Sadowski, John Salvatier, François Savard, Jan Schlüter, John Schulman, Gabriel Schwartz, Iulian Vlad Serban, Dmitriy Serdyuk, Samira Shabanian, Étienne Simon, Sigurd Spieckermann, S. Ramana Subramanyam, Jakub Sygnowski, Jérémie Tanguay, Gijs van Tulder, Joseph Turian, Sebastian Urban, Pascal Vincent, Francesco Visin, Harm de Vries, David Warde-Farley, Dustin J. Webb, Matthew Willson, Kelvin Xu, Lijun Xue, Li Yao, Saizheng Zhang, and Ying Zhang. 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`.

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`.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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