Neurocomputing 393 (2020) 165–174

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

1.2 Watt Classification of 3D Voxel Based Point-clouds using a CNN on a Neural Compute Stick

Xiaofan Xu[a,*], Sam Caulfield[a], Joao Amaro[b], Gabriel Falcao[b], David Moloney[a]

[a] Movidius Division, Intel Ireland Limited, Ireland
[b] Dept. of Electrical and Computer Engineering, Instituto de Telecomunicações, University of Coimbra, Portugal

ARTICLE INFO

Article history:
Received 17 November 2017
Revised 19 September 2018
Accepted 28 October 2018
Available online 24 July 2019

Keywords:
3D Convolutional Neural Network
3D object classification
Embedded systems
Low-power

ABSTRACT

With the recent surge in popularity of Convolutional Neural Networks (CNNs), motivated by their significant performance in many classification and related tasks, a new challenge now needs to be addressed: how to accommodate CNNs in mobile devices, such as drones, smartphones, and similar low-power devices? In order to tackle this challenge we exploit the Vision Processing Unit (VPU) that combines dedicated CNN hardware blocks and very low power requirement. The lack of readily available training data and memory requirements are two of the factors hindering the training and accuracy performance of 3D CNNs. In this paper, we propose a method for generating synthetic 3D point-clouds from realistic CAD scene models to enrich the training process for volumetric CNNs. Furthermore, an efficient 3D volumetric object representation Volumetric Accelerator format (VOLA) is employed. VOLA is a sexaquaternary (power-of-four subdivision) tree-based representation which allows for significant memory saving for volumetric data. Multiple CNN models were trained and pruning techniques for the weights were applied to the trained 3D Volumetric Network in order to remove almost 70% of the parameters and outperform the existing state-of-the-art networks. The top performing and efficient model was ported to the Movidius™ Neural Compute Stick (NCS). After deployment on the NCS, it takes 11 ms (~90 frames per second) to perform inference on each input volume, with a reported power requirement of 1.2 W, which leads to 75.75 inferences per second per Watt.

1. Introduction

Recently, computer vision has made great advancements in 2D object recognition [1–4]. This is due to ImageNet [5], which provides millions of images for training these networks and achieve nearly human level performance. However, having an understanding of the 3D environment is an important aspect for computer vision researchers as well. Similar to the 2D object recognition problems, 3D data is an essential requirement for training 3D models. Real-time Simultaneous Localization and Mapping (SLAM) techniques and different products readily available in the market such as the Microsoft Kinect or Intel RealSense have increased the number of 3D datasets available to researchers [6–10]. In this paper, we use the 3D geometric data created by ourselves combined with the existing objects from ModelNet [10] and real-world scene in point-cloud format for 3D classification task.

Performing object recognition on 3D point-cloud occluded volumes depicting real-world scenes containing ubiquitous objects presents multiple challenges, including the significant memory requirements for these volumetric representations. Therefore, we converted the point clouds to a 3D sexaquaternary-based voxelized grid, to both minimize the memory footprint and to be able to present the data to the network in an efficient manner.

CNNs have achieved great performance in 2D object recognition challenges, naturally we extend 2D convolutions to 3D for our work. However, the computation requirement for 3D convolution is very complex in the volumetric domain. The sparsity in the 3D volumetric data can still introduce many unnecessary weights in the hidden layers. This can easily cause overfitting issues and will lead to millions of parameters in the end. Therefore, in our work we developed and implemented some sparsity techniques to the 3D convolutional network in order to reduce the parameters and allow the 3D network to achieve better generalization on 3D vision problems. Also we further reduce the size of our network from 3D to 2D projection similar to [11] and evaluate between different techniques. Furthermore, we discovered that most of the existing 3D networks cannot perform real-time recognition on any

* Corresponding author.
E-mail address: xu.xiaofan@intel.com (X. Xu).

https://doi.org/10.1016/j.neucom.2018.10.114
0925-2312/

embedded systems due to the size and speed of these networks and there is a lack of platforms capable of delivering inference at high FPS and at very low power per inference. In this work, we wish to make the move to real-time recognition of objects in mobile, power-constrained or autonomous systems (e.g., cars, drones, robots, etc.). We need to address a change of paradigm capable of coping high computational power with low-power supply requirements and low-energy consumption levels. Therefore, we use a recent released Movidius™Neural Compute Stick [12], which can perform CNNs in real-time at very low power.

The main contributions of this paper are as follows:

- A method for the synthetic generation of realistic 2.5D point-clouds with occlusion of the back-face of the objects. Taking the CAD models offered by the ModelNet10 dataset [10] and placing them randomly in a set of realistic room CAD models, we are able to generate realistic point-clouds. The 2.5D point-clouds are then converted to a 3D sexaquaternary-based voxelized grid, to both minimize the memory footprint, and to be able to present the data to the network in an efficient manner;
- Multiple CNN models are developed, supported by our modified ModelNet10 dataset, and a new sparsity technique is applied to the 3D network. Furthermore, the light weight model is ported to a very low-power and low cost Movidius™Neural Compute stick [12] based on Myriad 2 VPU [13].

2. Related work

Convolutional Neural Networks (CNNs) have been shown to be powerful classification tools for multiple real-world computer vision tasks, such as scene segmentation and labelling [14], medical imaging diagnosis [15,16] and object detection in autonomous driving [17], in many cases approaching near-human performance. The recent popular CNNs [3,18,19] are focusing on efficiency as well as accuracy. With 2D networks running low-power and less memory, deep learning on 3D data also need to move towards the trend.

The concept of 3D networks has been used in many areas such as hand gesture recognition [20] and human action recognition [21]. The authors in [10] (3D Shapenets) created the publicly available ModelNet 3D dataset, which contains 151,128 CAD models, distributed over 660 categories. Like most of the CNN work on 3D data, we choose 10 categories from the dataset, and compute the corresponding 2.5D point-cloud. In [10] work, the authors use 3D voxel grid as input to the CNN with 3D kernels for CNN. This work was the first to apply CNN on 3D representation. Similar to their approach, VoxNet [22] also used the voxelized 3D input to the CNN. The advantage of these approaches is that they can take in 3D data in different format including LiDAR, point-clouds or CAD models. Following the success of these approaches, we pursued the same direction.

However, due to complexity and the high amount of parameters necessary for performing 3D convolution, a multi-view CNN (MVCNN) [23] was developed. The authors use multiple rendered views of 3D data, passed the input as 2D images. The CNN was trained and process the views jointly. MVCNN was pre-trained on ImageNet [5], and achieve satisfied accuracy on the ModelNet dataset. The issue for this approach is that it requires a separate pipeline for generating the render images before training the CNN. The authors in [11] solved this issue using 'X-ray scanning' process insider the CNN model which means the input for the model is still a 3D voxel grid. Instead of the render images, in [11] work, the network can take 3D cubes and aggregate information to 2D planes. In our work, we further improved their algorithm with a smaller amount of parameters.

Apart from the above methods, there are a lot of new ideas coming out recently in the literature to work with 3D object recognition. PointNet [24] is the first work to directly use the point-cloud coordinates (i.e., only use the (x, y, z) coordinate as the points channels) as input to the network. [25] further improves PointNet with points local neighbourhood, they proposed 2 operations focus on local 3D geometric structures and local high-dimensional features. Furthermore, [26] proposed a new approach by using encoder-decoder structure based on Recurrent Neural Networks (RNNs) with attention for learning the 3D global features and achieves promising accuracy on ModelNet dataset. However, most of these methods do not consider the heaviness caused by weights in the network. Moreover, 3D data is sparse in nature, by using normal convolutional layers causes lots of redundant weights appearing in the kernels.

Although large datasets training and validation in clusters or servers are important [27–29], but in the low power end people have increase concern about efficiency for CNNs, works like [30,31] focused on quantization and sparsity for 2D CNNs in order for these models to fit onto embedding systems or mobile devices. Network optimization (i.e., network pruning or quantization) is a technique to reduce the model size by compressing the dense model into sparse or low-bit architecture with minimal or even no accuracy drop. To be more specific, Guo et al. [32] shows compress parameters by a factor of 17.7 × on AlexNet resulting no accuracy drop. For 3D CNN networks, OctNet [33] took the advantage of using sparse data and developed an efficient way for implementing convolutional layer. However, exclude the implementation in OctNet, quantization and sparsification have not been fully explored in the 3D CNNs. In our work, we focus on sparsification and we use similar approach as [34] by finding redundant connections in the network first, and use the remaining connections in the fine-tuning stage to recover the accuracy. To be more specific, we have performed network sparsification which can remove almost 70% of the weights in the original 3D CNN. Moreover, our sparse network is able to run with any standard frameworks without any modification. This means our pruned model can be easily transfer onto any embedded systems with sparse matrix multiplication support and achieve instant speed-up compare to the original 3D CNN.

3. Sexaquaternary tree based dataset

3.1. Volumetric data structure

This work uses an adapted version of the octree to represent the 3D data. The hierarchical grid-based data structure is a bit-per-voxel sexaquaternary (64-ary) tree show in Fig. 1. It provides several benefits:

- Compatibility with the output of the Blensor algorithm [35]: the output of Blensor is a point-cloud where the points represent an approximation of the surfaces in the scene. The points can be easily converted into a voxel data structure using "binning". The voxel data structure is a regular 3D grid of homogoneous axis-aligned cubes. Each cube has an (x, y, z) position where x, y and z are natural numbers only. Using
$$p' = (p - t) \times s \tag{1}$$
where p' is the transformed point, p, t is the position of the minimum point in the cloud and s is the ratio between the point-cloud bounds and the voxel volume bounds, points from the point-cloud can be transformed into "voxel space", i.e. a position in the voxel volume;
- Fast dataset noise reduction using simple grid-based nearest neighbour elimination: the above technique implicitly performs a noise reduction step around surfaces by "binning" clusters of points into the same voxel. This can simplify the dataset considerably by reducing the number of points while still retaining the general surfaces;

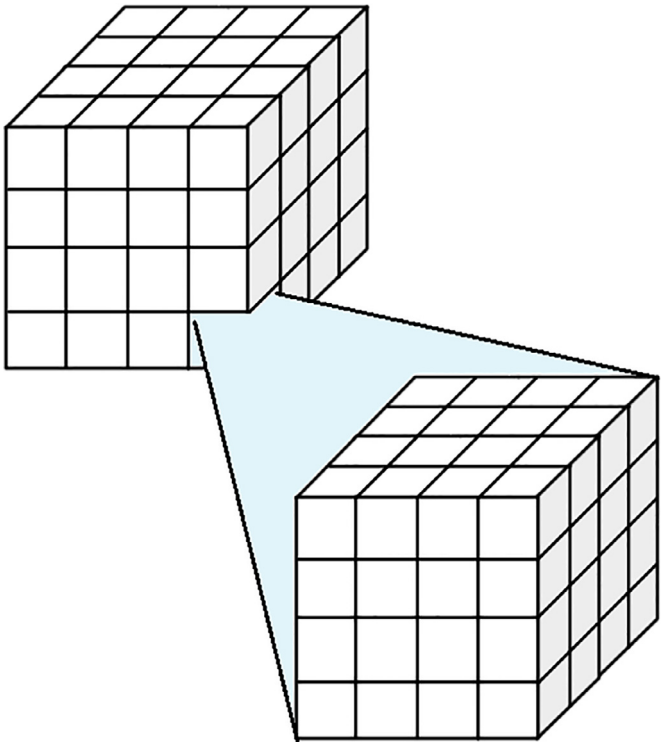


Fig. 1. Each voxel at level N spatially encodes 64 voxels at level N+1.

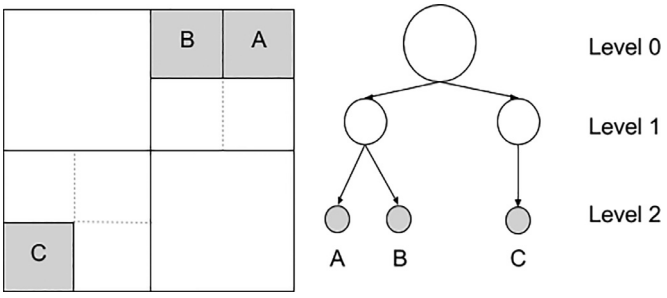


Fig. 2. A simple example of a quadtree encoding pixels (the 2D equivalent of a voxel). This same technique is employed in 3D with higher-order trees.

- Fast voxel data compression using implicit level-of-detail (LOD): the voxel sexaquaternary tree automatically populates itself at $log(4, N)$ levels, where N is the side length of the volume in voxels. Each level in the sexaquaternary tree is a subsampled version of the next level (as exemplified in Fig. 2), and as a result is smaller in memory and faster to access because there is less logic required to access data in higher levels of the tree.

Using the voxel sexaquaternary tree representation, smaller representations of the data were generated while maintaining object shape, relative scale, and curtailing detail uniformly which can be seen in Fig. 3. The tree representation of the data is generated using a simple one-pass algorithm that processes the point-cloud generated by KFusion. The algorithm discretises the point-cloud data set into axis-aligned, isomorphic three-dimensional bins (voxels) that may contain multiple points, reducing the size of the dataset while maintaining dataset quality to a given level of detail. The voxel sexaquaternary tree implicitly constructs lower LOD representations of the scene geometry. An important consequence of this feature of the data structure is that it facilitates multi-scale classification. These lower LOD representations are encoded as higher levels in the data structure. The low-LOD representation

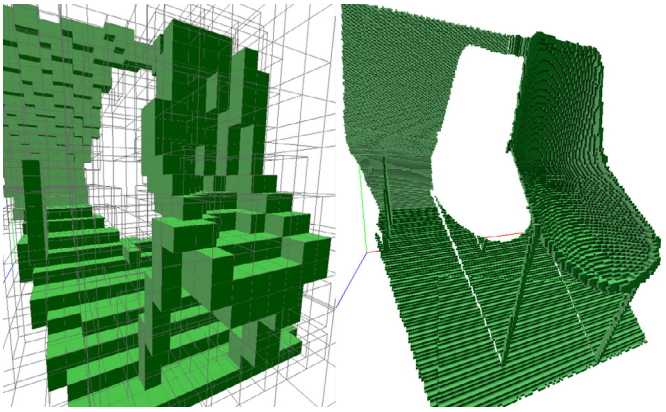


Fig. 3. Full level ($256 \times 256 \times 256$ occupancy grid on the right) vs. low level ($32 \times 32 \times 32$ occupancy grid on the left) of details for a chair taken from ModelNet10 [10] with background in voxelized point-cloud format.

reduces the size of the data by several orders of magnitude (in base-4 in the case of the sexaquaternary tree). This allows significant memory savings for volumetric data. A key advantage of this tree implementation over the octree is that the levels of detail are more readily useful. In an octree, the first level contains eight voxels, and the resolution of each successive level is twice that of the previous level. In the tree used in this work, the resolution of the first level is 64 voxels which increases by a factor of 64 at each level. The purpose of this is to avoid having multiple redundant low-resolution levels of detail because the first few levels in an octree may be too low-detail to be useful in some applications. We call this voxelized representation VOLA (Volumetric Accelerator) in this paper. Furthermore, VOLA uses a one bit per voxel format to compress the volume contents. The intended use case for one bit per voxel representation is representing occupancy, i.e., indicating if a voxel is either completely solid or completely empty.

Once the point-cloud has been converted into the voxel representation, the sparse voxel sexaquaternary tree can be exported as a sequence of ones and zeros representing the voxel occupancy values in the volume. We reshape the single sequence into a 3D volume that contains bit values and pass it as an input to the network. This binary format can save computational cost in the initial layer of the network.

3.2. Voxel Based Point-cloud

The dataset is generated inside Blender using a plug-in called Blensor [35]. We load the objects into Blender and using a simulated Kinect Sensor in Blensor simulate the point-clouds. In order to generate the voxelized objects for the CNN, we converted the point-clouds into VOLA format to achieve binary inputs which contain only ones and zeros.

Our dataset contains objects with background in point-clouds and voxelized formats which make it differ from the existing datasets. The dataset is captured using Time of Flight (TOF) camera which contains sensors to sense the time that it takes light to return from any surrounding objects in order to measure the distance between a sensor and an object. The reason for adding background into our objects is because when using TOF camera such as Intel RealSense[1] or Kinect to capture the point-clouds in real world scenarios, we found that the background scene would always appear in the simulated data. The 3D objects in our dataset are taken from ModelNet10 [10] and 3D models for the rooms which are

[1] http://www.intel.com/content/www/us/en/architecture-and-technology/realsense-overview.html.

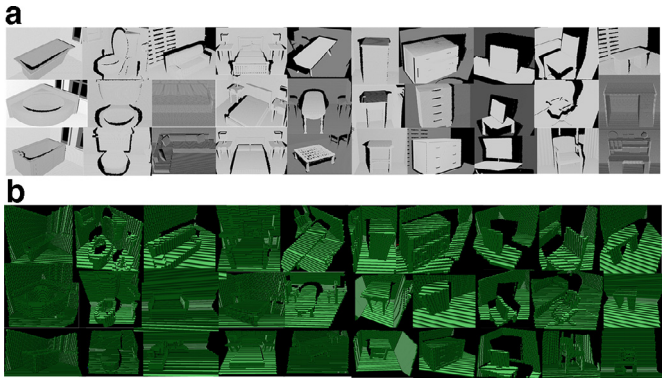


Fig. 4. ModelNet10 [10] objects placed in different rooms with distinct formats: (a) Point-clouds contain objects with background, which are captured in Blender using Kinect sensor as TOF camera; (b) Point-clouds are converted into voxelized representations using VOLA, where each voxel is represented as 1 and the empty space is represented as 0.

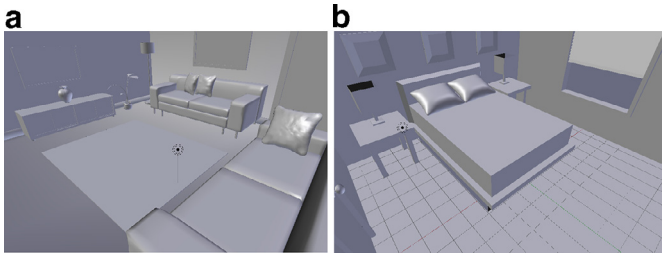


Fig. 5. Examples of 3D scenes [36] which are used in our dataset as a background for the objects: (a) living room scene; (b) bedroom scene.

used for the background scenes are taken from [36]. Fig. 5 shows some sample rooms which were used for our dataset. We created the dataset by placing objects randomly inside different 3D rooms taken from [36]. The whole process takes place inside Blender. The TOF camera in Blender automatically faces towards the object and acquires the point-cloud using the Kinect sensor from Blensor. The acquired point-clouds from the Kinect sensor inside Blender are shown in Fig. 4(a). We then converted the point-cloud into voxelized VOLA format shown in Fig. 4(b). In this case a fixed occupancy grid of size $64 \times 64 \times 64$ is used in order to maintain the useful information from the point-clouds. The total number of 3D voxelized point-clouds for training and testing are 3991 and 908 respectively, which is the ModelNet10 distribution.

4. Convolutional Neural Networks on volumetric data

4.1. Overview

The successful results achieved using CNNs on 3D objects [22,33,37–40] and etc. encourage us in that direction. CNNs can explicitly encode spatial structures of the inputs, i.e. planes and corners of the 3D objects associated with different orientations and positions. In addition, CNNs have the ability to stack multiple layers in order to conduct a hierarchy including complex features regarding 3D regions and finally lead to a global label for the 3D input. Furthermore, a trained CNN model can be easily deployed to a hardware platform to perform inference with only a feed-forward pass which is very efficient for classification purposes.

In this work, we introduce 2 different CNNs, which both take in inputs as explained in Section 3. Both networks can achieve state-of-the-art results on 3D volumetric data. The first network is inspired by Maturana and Scherer [22] and further pruning technique was applied on this network as well. This network learns the 3D information from the volumetric data as it uses 3D kernels which can store features including orientations, positions and depth information of the 3D volumetric data. The second network we developed is a light weighted network which is very suitable for embedded devices and we have ported this network onto Neural Compute Stick [12] which we explain in detail in later sections. This network takes in 3D volumetric data, which makes it different from the exiting work [23]. Instead of rendering different views of the 3D input in computer graphics, it projects the 3D shape to 2D planes by convolving its 3D volume using 2D kernels and classification is transformed on these 2D plans. This process is end-to-end with standard layers in CNN and achieves similar accuracy compare to the 3D Volumetric Network on the proposed dataset.

4.2. Network 1: 3D Volumetric Network

The 3D Volumetric Network uses 3D kernels to extract features in 3D space. Different from the exiting networks, our network is much simpler and performs better on a more complex dataset. In order to cope with overfitting issues on the training data shown in [10], we also introduce a pruning technique that can increase the model capabilities on the test data and make it more robust.

4.2.1. 3D Volumetric Network layers

In order to conduct the proposed network, we use the following layers.

Input layer. This layer is the first layer for both networks which accepts an occupancy grid of fixed size $X \times Y \times Z$. In our work, $X = Y = Z = 64$. With the VOLA representation explained in Section 3, the input value for our data is either 0 or 1, which means multiplications are trivial, i.e., multiply by 0 produces 0 and multiply by 1 propagates the weight to the output, so no multiplications are required. There is no further pre-processing requirement for the data. This means our input data contains only 1 bit per voxel in order to represent the 3D volume.

3D Convolutional layer. This is the most important layer in the proposed 3D CNN. For a single 3D convolution kernel with weights $W \in \mathbb{R}^{L \times M \times N}$ convolving with a 3D tensor T^{in}, the output tensor T^{out} at position (i, j, k) is given as

$$T^{out}_{i,j,k} = \sum_{l=0}^{L-1} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} W_{l,m,n} T^{in}_{i+l,j+m,k+n} \tag{2}$$

We call the output 3D tensor in our work feature block, which contains the result of 3D convolutions being performed. A spatial stride s is also applied to the convolution.

A Rectified Linear Unit (ReLU) activation function is followed after each convolution which performs nonlinearities in networks to increase the model's capabilities and prevents a vanishing gradient problem during back propagation of the network. The ReLU activation function is given as:

$$T^{out} = ReLU(T^{in}) = max(T^{in}, 0) \tag{3}$$

Fully connected layer. This layer can produce n output neurons and the output of each neuron is a learned linear combination of all the output neurons from the previous layer. In our work n is 10. Different from the convolutional layer, this layer is not spatially located. This layer is used for both networks as well.

Softmax layer. This layer is usually used for multiclass classification tasks. The sum of the output from softmax layer is always 1, which means it squeezes the output between 1 and 0. This layer gives a probability result which can highlight the maximum value from the previous layer's output and minimize the value which is significant below the maximum value. The representation for each output element T^{out}_j in the softmax is shown in Eq. (4). In our

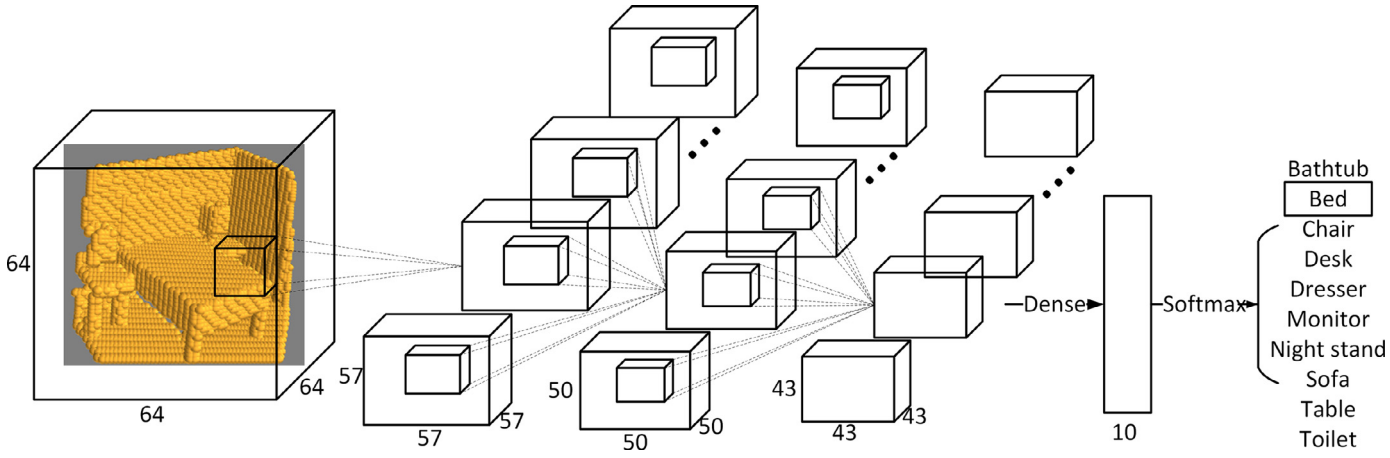


Fig. 6. 3D CNN architecture layout for objects with background. The input is $64 \times 64 \times 64$ with VOLA format with each voxel represented by 1 bit. The input passes through 3 convolutional layers and 1 fully connected layer. The kernels used for convolutional layers are $8 \times 8 \times 8$ with stride 1 which gives output feature blocks with size 57^3, 50^3 and 43^3 from Conv1, Conv2 and Conv3 respectively. The output from fully connected layer is 10 units.

work, the range for j is between 0 and 9, as we have 10 different categories.

$$T_j^{out} = \sigma(T_j^{in}) = \frac{e^{T_j^{in}}}{\sum_{n=1}^{N} e^{T_n^{in}}} \tag{4}$$

4.2.2. 3D Volumetric Network configuration

In order to train the network on our 3D objects with scene, we have developed a volumetric architecture which contains the layers explained in Section 4.2.1. Fig. 6 represents the 3D Volumetric network architecture for our dataset. The input to the network is $64 \times 64 \times 64$ as described in Section 3. There are 3 convolutional layers, 1 fully connected layer and a softmax layer to produce the output. We also add a regularization layer which is dropout layer after the third convolutional layer with drop rate of 0.5 in order for the network to generalize better [41]. The 3D kernel we used for the convolutional layers are $8 \times 8 \times 8$ which means in Eq. (2), $L = M = N = 8$ are used in our architecture. Based on Eq. (2), the first convolutional layer produces 8 feature blocks with size 57^3, the second and the third convolutional layers produce 16 feature blocks with size 50^3 and 43^3 respectively. After that we add 1 fully connected layer to produce 10 units and the final output gives the probability for different classes based on the input. This 3D CNN model is designed and trained using Caffe [42]. The learning rate for training the networks is 0.0001, the weight decay is 0.0005 and the momentum is 0.9.

4.2.3. 3D Volumetric Network Pruning

The complexity of our 3D CNN makes it containing millions of parameters. This means the network can easily overfit on the training data. Therefore, we have added more regularization and increased the robustness of the network by pruning the unnecessary weights during training which requires the following steps.

- Sort all the existing weights from maximum to minimum;
- Choose a threshold to prune all the weights below this threshold, i.e., set to zero;
- Retrain the network with the weights from the previous step. No update would perform on the weights that are set to 0 during the retraining, i.e., weights set to 0 will remain 0 during training.

Similar works done by Zhou et al. [30], Guo et al. [32] and Han et al. [34] show great results in 2D CNNs. In our work, we extended it to 3D CNNs. We analysed weights in the original trained 3D CNN and have found that most of the weights are very small. The 3D kernels contain weights in the first layer of 3D convolution is shown in Fig. 7(a). Each circle contains the weight value in the 3D kernel. Red, orange and yellow weights are more important than green, cyan and blue ones. The bigger the circle, the bigger the weight value. After observing the 3D kernels, we decided to prune 70% of the weights in each layer as shown in Table 1. After retraining the model, it maintained the same accuracy after pruning, which shows that the zero weights occupied the most of the 3D kernels. We extracted the weights in the same layer (Conv1) from the pruned trained model shown in Fig. 7(b). The 3D weight kernels in the pruned model are much more sparse than the previous weight kernels before pruning. As in 3D volume, input data contains a huge volume of empty space, therefore weights associated with these spaces should not have important values, as no feature can be learn from empty spaces.

Table 1
Number of zeros in each layer of the 3D CNN after compression.

Layers	No. of parameters	No. of zeros
Conv 1	$8 \times 1 \times 8 \times 8 \times 8$	2867
Conv 2	$16 \times 8 \times 8 \times 8 \times 8$	45,875
Conv 3	$16 \times 16 \times 8 \times 8 \times 8$	91,750
FC	10×1272112	8,904,784

4.3. Networks 2: 3D-to-2D Projection Network

Recent successes in multi-view CNNs [23] project the 3D objects to 2D and then use well-developed 2D image's CNNs for classification. However, the projection stage uses the external rendering pipelines from computer graphics. In [11], the authors proposed a 3D-to-2D projection using anisotropic probing which is capable of capturing internal structures of objects through a manner which is similar to 'X-ray scanning'. The success of these approaches inspired us to introduce a much simpler and more lightweight network containing only existing layers in 2D CNNs as shown in Fig 8. Note that our input to the network is still 3D, which uses the same input layer as the 3D Volumetric Network in Section. 4.2.1.

The nature of our dataset is a point-cloud format, thus 1 out of the 3 dimensions does not store as much information as the other 2. We have found that 2D convolutional layers are capable of capturing the internal and global structure of the input volume through its end-to-end projection base on our data. This network

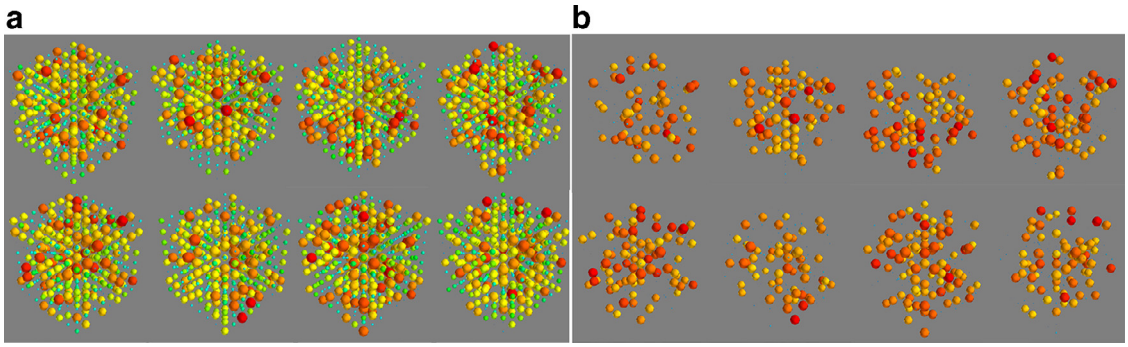


Fig. 7. 3D kernels in the 1st convolution layer of the model. Red, orange and yellow weights are more important than green, cyan and blue weights. The bigger the circle, the bigger the weight value. (a) 3D kernels before pruning; (b) 3D kernels after pruning.

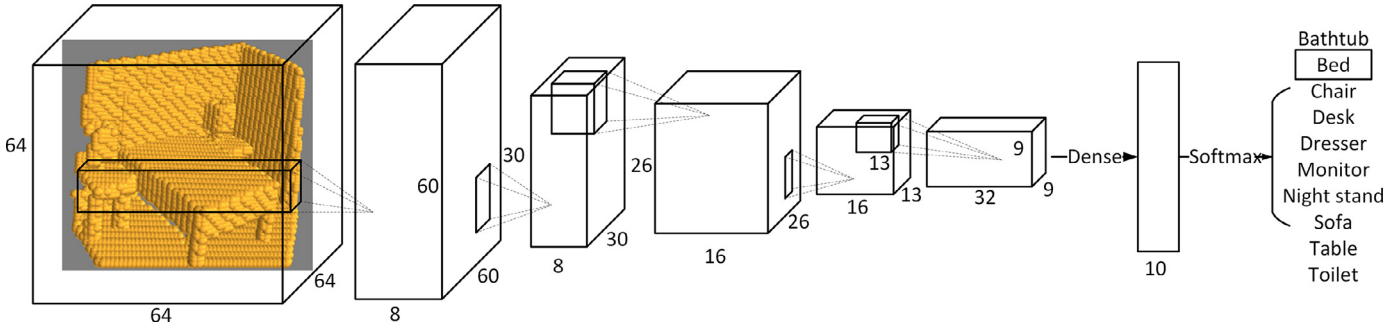


Fig. 8. 3D-to-2D Projection Network architecture layout for objects with background. The input is $64 \times 64 \times 64$ with VOLA format and each voxel represented as 1 bit. Input passes through 3 convolutional layers followed by pooling layers with kernel size 2×2 and stride of 2. Fully connected layer is followed which is the last layer of the network. The kernels used for convolutional layers are 5×5 with stride 1. The output from fully connected layer is 10 units for representing 10 different classes.

can learn the early features with much less parameters and the features it learns cannot be achieved by standard rendering. The simplicity of this model means it is very easy to deploy onto any embedded device, in our case in a Neural Compute Stick introduced in Section 6.

4.3.1. 3D-To-2D Projection Network configuration

In contrast to the traditional 3D networks [10,22,23] our 3D-to-2D Projection network uses 3 convolutional layers with kernel size 5×5, which can aggregate long-range interactions. For the first 2 convolutional layers, each of them is followed by a max-pooling layer and a nonlinearity linear layer ReLU. The max-pooling layers with kernel size 2×2 are used to output the most significant feature inside each kernel. The third convolution layer is directly connected to the fully connected layer to produce 10 units as explained in Section 4.2.1 followed by a softmax layer gives the probability for the different classes based on the input. This model is trained using the same configuration as the 3D Volumetric Network, but it takes more iterations to converge in order to achieve similar accuracy.

5. Experimental results

5.1. Performance based on accuracy evaluation

The networks proposed in Section. 4 are trained on 3991 3D voxelized point-clouds and tested on 908 3D voxelized point-clouds in this section and further tested on occluded point-clouds for all the classes in Section. 5.3 without any fine-tuning. The input occupancy grid is 64^3 with VOLA format both for training and testing. To be more specific, the input contains only 1 bit per voxel and no further pre-processing operation is applied to the input.

In order to evaluate our network's performance, we compare our method with the state-of-the-art shape classification networks on the ModelNet10 dataset. The overall comparison is shown in Table 2. However, our dataset is more complex than ModelNet10 dataset as it contains both objects and a background scene.

Table 2
Accuracy achieved for different existing CNN models and our models. The existing CNN models are trained and tested on pure ModelNet10 objects where our networks are trained and tested on voxelized point-clouds containing ModelNet10 objects with background.

Dataset	Network	Acc
ModelNet10	PointNet [43]	77.6%
	3D ShapeNet [10]	84%
	OctNet [33]	90.1%
	VoxNet [22]	92%
ModelNet10 + background (Ours)	3D-to-2D Projection Network	91.7%
	3D Volumetric Network	93.5%
	Pruned 3D Volumetric Network	94.1%

3D Volumetric Network. We tested the network before pruning and after pruning using the same test data. The average accuracy after pruning the network is slightly higher than before pruning. Similar performance is shown in [30] on 2D CNNs. After quantizing and pruning the weights, most of the existing 2D CNNs achieve slightly better accuracy. The pruning technique can prevent the network from overfitting as it added more regularization and pruned unnecessary weights in the network as explained in Section 4.2.3. The confusion matrix for the network before pruning is shown in Table 3. Bathtub performs the best which achieves 100% and dresser has the lowest accuracy at 73%. The confusion for dresser is mostly between the desk as they are very similar in nature. Overall, the 3D Volumetric Networks (before and after pruning) achieve higher accuracy than all the existing 3D networks on ModelNet10.

Table 3
Confusion matrix for trained 3D Volumetric Network on ModelNet10 objects with background. Darker background means higher percentage.

	bathtub	bed	chair	desk	dresser	monitor	night stand	sofa	table	toilet
bathtub	**1.00**									
bed		**0.98**	0.01			0.01				
chair		0.01	**0.97**		0.01					0.01
desk			0.01	**0.86**	0.05			0.08		
dresser			0.09	0.14	**0.73**			0.01		0.02
monitor						**0.99**		0.01		
night stand		0.01			0.01		**0.97**	0.01		
sofa					0.01			**0.97**		0.02
table			0.02						**0.98**	
toilet			0.01		0.04	0.05				**0.90**

Table 4
Confusion matrix for trained 3D-to-2D Projection Network on ModelNet10 objects with background. Darker background means higher percentage.

	bathtub	bed	chair	desk	dresser	monitor	night stand	sofa	table	toilet
bathtub	**1.00**									
bed		**0.98**				0.01				0.01
chair		0.01	**0.99**						0.01	
desk			0.01	**0.86**	0.04			0.09		
dresser			0.10	0.08	**0.72**	0.01	0.02	0.01		0.06
monitor						**1.00**				
night stand							**0.99**			0.01
sofa					0.01			**0.97**		0.02
table			0.03	0.01			0.01		**0.95**	
toilet			0.07			0.02	0.18			**0.73**

Table 5
Number of parameters in the existing networks. The "M" stands for million.

Dataset	Input size	Network	#params
ModelNet10 + background (Ours)	$64 \times 64 \times 64$	3D Volumetric Network	12 M
	$64 \times 64 \times 64$	Pruned 3D Volumetric Network	9 M
	$64 \times 64 \times 64$	3D-to-2D Projection Network	0.05 M
ModelNet10	$32 \times 32 \times 32$	VoxNet [22]	1 M
ModelNet40	$30 \times 30 \times 30$	Subvolume [11]	16.6 M
	$224 \times 224 \times 3$	MVCNN [23]	6 M
	# of pts	PointNet [24]	3.5 M

3D-to-2D Projection Network. This light weight network achieves overall accuracy of 91.7% on 3D test data. It is slightly lower than the 3D Volumetric Networks introduced earlier but higher than the existing networks except VoxNet. The confusion matrix is shown in Table 4. Bathtub achieves 100% while dresser, desk and toilet are performing poorly. Note that this network stores much less parameters compared to the other networks.

5.2. Performance based on time and space

The networks in this work are compared to the existing networks on 3D data with memory requirement. Table 5 summarizes the number of parameters in the networks. While Subvolume [11], PointNet [24] and MVCNN [23] achieves high accuracy on their datasets, our 3D-to-2D Projection Network and Pruned 3D Volumetric Network offers a much smaller memory requirement. 3D-to-2D Projection Network is 70 × smaller than PointNet, 332 × smaller than Subvolume and 1200 × smaller than MVCNN. The pruned 3D Volumetric network is 1.8 × smaller than Subvolume and 6.7 × smaller than MVCNN. Although VoxNet has less parameters compared to the 3D Volumetric Networks that we proposed in this work, our input data is 8 × bigger than VoxNet inputs and our 3D-to-2D Projection Network is still 20 × smaller.

Our 3D-to-2D Projection Network can perform 0.5*ms*/*inference* using a Titan X GPU on Caffe, and 13 ms/*inference* on Intel i7. This shows great potential for the network to run in real-time applications. The amount of parameters it requires makes it incredibly easy to deploy onto any embedded device.

5.3. 3D Volumetric Network on occluded point-clouds

Furthermore, we have analysed the ability of our trained 3D Volumetric Network for handling different occlusions in the point-clouds without any fine tuning or retraining involved. This can test the robustness of our trained 3D Volumetric Network. We use 3 directions to occlude the object with background in a 3D volume

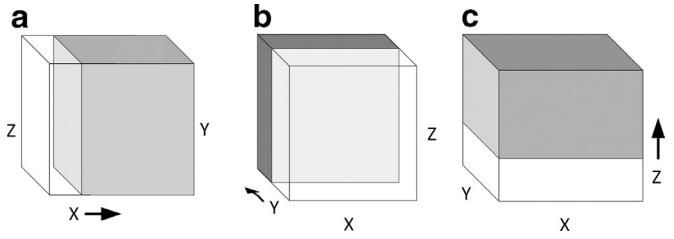


Fig. 9. Occlusion of the point-clouds in 3 directions inside the 3D volume. (a) Occlude the point-clouds in x direction with grey volume as occupied volume. (b) Occlude the point-clouds in y direction with grey volume as occupied volume. (c) Occlude the point-clouds in z direction with grey volume as occupied volume.

Table 6
Accuracy achieved for our trained 3D Volumetric Network based on the voxelized point-clouds with different percentage of occlusion in x, y and z directions.

	% of occlusion	mean Acc
Baseline	0%	93.5%
x-direction	20%	91.1%
	30%	86.8%
	40%	74.5%
y-direction	20%	88.6%
	30%	86.1%
	40%	83.3%
z-direction	20%	91.3%
	30%	85.4%
	40%	81.1%

Fig. 10. Movidius™ Neural Compute Stick based on Myriad 2 VPU which can load trained CNN architectures and perform inferences on it. This USB device is low-power with only a 1.2 W requirement.

of 64^3. Fig. 9 demonstrates the methods. We have considered different percentage of occlusions associated with each direction, i.e., 20%, 30% and 40%. The accuracies achieved for occluded voxelized point-clouds using our 3D CNN are shown in Table 6. It clearly shows that our 3D CNN can easily handle occlusions up to 20% in both x and z directions with slight loss in average accuracy compared to the original point-clouds. Although for some objects the occlusions make them difficult for recognition i.e. confusion among desks, tables, dressers and chairs. Overall, our 3D CNN can cope up to 40% of occlusion on the voxelized point-clouds.

6. CNN on Neural Compute Stick

The trained 3D-to-2D Projection Network has been imported to the Neural Compute Stick (NCS) [12] shown in Fig. 10, which can be used on low-cost robotics or drones and perform navigation based on the inference. This USB device can offload compute-intensive Deep Neural Networks while still demanding low-cost and low-power. The network performance on the NCS is shown in Section 6.2.

6.1. Intel® Movidius™ Neural Compute Stick

The NCS is a low-cost and low-power USB device based on Myriad 2 VPU [13]. It supports loading networks designed and trained on common deep-learning frameworks such as TensorFlow [44] and Caffe [42]. Any AI programming can run at the edge by using NCS. The NCS combined with Movidius Neural Compute SDK allows a deep learning developer to profile, tune, validate and deploy CNNs on low power applications that require real-time inference.

Table 7
Performance of 3D-to-2D Projection Network on different hardware platforms with respect to classification time, power and energy consumption. (* higher is better).

Platform	Nvidia Titan X	NCS	Intel Core i7-5930K @ 3.50 GHz
Time (ms)	0.5	11	13
Power (W)	250	1.2	95
Inference/time/power (inference/sec/W)*	0.8	75.75*	0.81

6.2. Network inference on Neural Compute Stick

The 3D-to-2D Projection Network is trained and deployed on the NCS. The input in this case is still 64^3. The network architecture is exactly the same as explained in Section. 4.2.2 which achieved 91.7% on the test data. We analyse the performance of the network by looking at the inference time, as we wish to run the 3D classification in real-time. The average run-time for the 3D-to-2D Projection Network to perform over 1000 inferences on 64^3 voxelized point-clouds using NCS powered by Myriad 2 VPU is only 11 ms with 12 Streaming Hybrid Architecture Vector Engines (SHAVEs). In order to compare the speed of the network running on different platforms, i.e., CPU, GPU, we tested the network running on:

- VPU: Intel® Movidius™ Neural Compute Stick;
- CPU: Intel Core i7-5930K @ 3.50GHz, 32G RAM;
- GPU: Nvidia GeForce GTX Titan X.

The result shows that network running on NCS is faster than the inference performed on Intel Core i7 and 22 times slower than Nvidia Titan X, as shown in Table 7. Moreover, inference per second per Watt is analysed using *inference÷time÷power* given the power for all 3 platforms shown in Table 7. Power requirements for NCS is analysed in Section. 6.3. NCS powered by Myriad 2 performs 75.75 inferences/Sec/Watt. The energy consumption results displayed in Table 7 show the Intel Core i7 is 93.5 × worse than Myriad 2, and the Nvidia Titan X is 95 × worse than Myriad 2.

With NCS, we can specify the number of SHAVEs for classification purposes in order to achieve the optimal results. The run-time for the network will vary depending on the number of SHAVEs being used. Due to memory allocation and data transfer, with 8 SHAVEs this network achieves optimal run-time performance with an average time of 10.33 ms to perform 1 classification. By looking at the pre-layer profile statistics performance, nearly half of the classification time is spent on the first convolutional layer. This is because most of the calculations and most of the convolutions happen in the first layer.

6.3. Power requirements for network on NCS

The power usage for the trained network on NCS is measured during the inference stage. The setup is shown in Fig. 11, and as in [45] we use Raspberry Pi 3. NCS is attached to the Raspberry Pi over USB for offloading the model to perform inference. The power is visualized by using INA219 power monitor IC from Texas Instruments attached to the NCS's USB connection directly. The average powerachieved is 1.235 W for the network. The peak power usage happens at the convolutional stage and is less than 2 W.

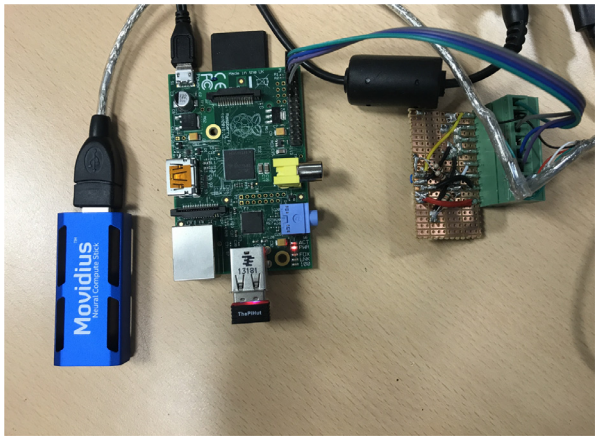


Fig. 11. Setup used for evaluating the power needs of NCS during inference time.

7. Conclusions

We presented two different network architectures in this work for 3D volumetric object recognition. We introduced a realistic 3D voxelized point-cloud dataset containing ModelNet10 objects in order to train the 3D CNN. The voxelized dataset contains only 1 bit per voxel using our VOLA algorithm, which saves computation in the first layer of the network. We enable further improvement of our 3D Volumetric Network by combining the technique introduced in the low precision networks in order to prune the weights in all the layers and reduce the computational cost. In addition, we analysed the performance of our networks based on accuracy and memory requirements where they achieve state-of-the-art results. Furthermore, the 3D-to-2D Projection Network based on our 3D voxelized point-clouds as input has been deployed onto a very low power NCS to perform inference. This means our network can be easily deployed in robotics, drones, cars and a variety of other low-power embedded systems to perform real-time navigation and classification.

Declarations of interest

None.

Acknowledgements

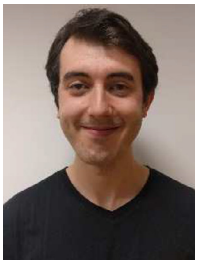
This work was partially supported by Fundação para a Ciência e a Tecnologia (FCT) and Instituto de Telecomunicações under grants UID/EEA/50008/2019 and PTDC/EEI-HAC/30485/2017.

References

[1] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 1–9.
[2] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.
[3] F.N. Iandola, S. Han, M.W. Moskewicz, K. Ashraf, W.J. Dally, K. Keutzer, Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size, arXiv:1602.07360 (2016).
[4] G. Huang, Z. Liu, L. Van Der Maaten, K.Q. Weinberger, Densely connected convolutional networks, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 4700–4708.
[5] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A.C. Berg, L. Fei-Fei, ImageNet large scale visual recognition challenge, Int. J. Comput. Vis. (IJCV) 115 (3) (2015) 211–252, doi:10.1007/s11263-015-0816-y.
[6] N. Silberman, D. Hoiem, P. Kohli, R. Fergus, Indoor segmentation and support inference from RGBD images, in: Proceedings of the Computer Vision–ECCV, 2012, pp. 746–760.
[7] S. Song, S.P. Lichtenberg, J. Xiao, Sun RGB-D: a RGB-D scene understanding benchmark suite, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 567–576.
[8] J. Xiao, A. Owens, A. Torralba, SUN3D: a database of big spaces reconstructed using SfM and object labels, in: Proceedings of the IEEE International Conference on Computer Vision, 2013, pp. 1625–1632.
[9] A.X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, et al., Shapenet: an information-rich 3d model repository, arXiv:1512.03012 (2015).
[10] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, J. Xiao, 3d shapenets: a deep representation for volumetric shapes, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 1912–1920.
[11] C.R. Qi, H. Su, M. Nießner, A. Dai, M. Yan, L.J. Guibas, Volumetric and multi-view CNNs for object classification on 3D data, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 5648–5656.
[12] Intel®Movidius™, Intel® movidius™neural compute stick, 2017, (https://developer.movidius.com).
[13] B. Barry, C. Brick, F. Connor, D. Donohoe, D. Moloney, R. Richmond, M. O'Riordan, V. Toma, Always-on vision processing unit for mobile applications, IEEE Micro 35 (2) (2015) 56–66.
[14] A. Karpathy, L. Fei-Fei, Deep visual-semantic alignments for generating image descriptions, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 3128–3137.
[15] A. Prasoon, K. Petersen, C. Igel, F. Lauze, E. Dam, M. Nielsen, Deep feature learning for knee cartilage segmentation using a triplanar convolutional neural network, in: Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention, Springer, 2013, pp. 246–253.
[16] H.R. Roth, L. Lu, A. Seff, K.M. Cherry, J. Hoffman, S. Wang, J. Liu, E. Turkbey, R.M. Summers, A new 2.5 d representation for lymph node detection using random sets of deep convolutional neural network observations, in: Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention, Springer, 2014, pp. 520–527.
[17] C. Chen, A. Seff, A. Kornhauser, J. Xiao, Deepdriving: learning affordance for direct perception in autonomous driving, in: Proceedings of the IEEE International Conference on Computer Vision, 2015, pp. 2722–2730.
[18] A.G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam, Mobilenets: efficient convolutional neural networks for mobile vision applications, arXiv:1704.04861 (2017).
[19] J. Maria, J. Amaro, G. Falcao, L.A. Alexandre, Stacked autoencoders using low-power accelerated architectures for object recognition in autonomous systems, Neural Process. Lett. 43 (2) (2016) 445–458.
[20] P. Molchanov, S. Gupta, K. Kim, J. Kautz, Hand gesture recognition with 3d convolutional neural networks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, 2015, pp. 1–7.
[21] S. Ji, W. Xu, M. Yang, K. Yu, 3D convolutional neural networks for human action recognition, IEEE Trans. Pattern Anal. Mach. Intell. 35 (1) (2013) 221–231.
[22] D. Maturana, S. Scherer, Voxnet: a 3D convolutional neural network for real–time object recognition, in: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2015, pp. 922–928.
[23] H. Su, S. Maji, E. Kalogerakis, E. Learned-Miller, Multi-view convolutional neural networks for 3D shape recognition, in: Proceedings of the IEEE International Conference on Computer Vision, 2015, pp. 945–953.
[24] C.R. Qi, H. Su, K. Mo, L.J. Guibas, Pointnet: Deep learning on point sets for 3d classification and segmentation, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 652–660.
[25] Y. Shen, C. Feng, Y. Yang, D. Tian, Mining point cloud local structures by kernel correlation and graph pooling, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 4, 2018.
[26] Z. Han, M. Shang, Z. Liu, C.-M. Vong, Y.-S. Liu, M. Zwicker, J. Han, C.L.P. Chen, Seqviews2seqlabels: learning 3d global features via aggregating sequential views by RNN with attention, IEEE Trans. Image Process. 28 (2) (2018) 658–672.
[27] Y. You, Z. Zhang, C.-J. Hsieh, J. Demmel, K. Keutzer, Imagenet training in minutes, in: Proceedings of the 47th International Conference on Parallel Processing, ACM, 2018, p. 1.
[28] M. Duan, K. Li, X. Liao, K. Li, A parallel multiclassification algorithm for big data using an extreme learning machine, IEEE Trans. Neural Netw. Learn. Syst. 29 (6) (2017) 2337–2351.
[29] C. Liu, K. Li, K. Li, A game approach to multi-servers load balancing with load-dependent server availability consideration, IEEE Trans. Cloud Comput. (2018).
[30] A. Zhou, A. Yao, Y. Guo, L. Xu, Y. Chen, Incremental network quantization: towards lossless CNNs with low-precision weights, arXiv:1702.03044 (2017).
[31] M. Rastegari, V. Ordonez, J. Redmon, A. Farhadi, Xnor-net: Imagenet classification using binary convolutional neural networks, in: Proceedings of the European Conference on Computer Vision, Springer, 2016, pp. 525–542.
[32] Y. Guo, A. Yao, Y. Chen, Dynamic network surgery for efficient DNNs, in: Proceedings of the Advances In Neural Information Processing Systems, 2016, pp. 1379–1387.
[33] G. Riegler, A. Osman Ulusoy, A. Geiger, Octnet: Learning deep 3d representations at high resolutions, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 3577–3586.
[34] S. Han, J. Pool, J. Tran, W. Dally, Learning both weights and connections for efficient neural network, in: Proceedings of the Advances in Neural Information Processing Systems, 2015, pp. 1135–1143.
[35] M. Gschwandtner, R. Kwitt, A. Uhl, W. Pree, BlenSor: Blender Sensor Simulation Toolbox Advances in Visual Computing, in: Lecture Notes in Com-

puter Science, Springer, Berlin / Heidelberg, 2011, pp. 199–208, doi:10.1007/978-3-642-24031-7_20.
[36] A. Handa, V. Patraucean, V. Badrinarayanan, S. Stent, R. Cipolla, Understanding real world indoor scenes with synthetic data, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 4077–4085.
[37] X. Xu, D. Corrigan, A. Dehghani, S. Caulfield, D. Moloney, 3D object recognition based on volumetric representation using convolutional neural networks, in: Proceedings of the International Conference on Articulated Motion and Deformable Objects, Springer, 2016, pp. 147–156.
[38] X. Xu, A. Dehghani, D. Corrigan, S. Caulfield, D. Moloney, Convolutional neural network for 3D object recognition using volumetric representation, in: Proceedings of the First International Workshop on Sensing Processing and Learning for Intelligent Machines (SPLINE), IEEE, 2016, pp. 1–5.
[39] X. Xu, J. Amaro, G. Falco, D.M. Moloney, Classify 3D voxel based point-cloud using convolutional neural network on a neural compute stick, in: Proceedings of the IEEE Co-sponsored International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD), 2017.
[40] X. Xu, J. Amaro, S. Caulfield, A. Forembski, G. Falcao, D. Moloney, Convolutional neural network on neural compute stick for voxelized point-clouds classification, in: Proceedings of the 10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI), IEEE, 2017, pp. 1–7.
[41] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, J. Mach. Learn. Res. 15 (2014) 1929–1958.
[42] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, T. Darrell, Caffe: convolutional architecture for fast feature embedding, in: Proceedings of the 22nd ACM International Conference on Multimedia, ACM, 2014, pp. 675–678.
[43] A. Garcia-Garcia, F. Gomez-Donoso, J. Garcia-Rodriguez, S. Orts-Escolano, M. Cazorla, J. Azorin-Lopez, Pointnet: a 3D convolutional neural network for real-time object class recognition, in: Proceedings of the International Joint Conference on Neural Networks (IJCNN), IEEE, 2016, pp. 1578–1584.
[44] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al., Tensorflow: a system for large-scale machine learning., in: Proceedings of the OSDI, 16, 2016, pp. 265–283.
[45] D. Pena, A. Forembski, X. Xu, D. Moloney, Benchmarking of CNNs for low-cost, low-power robotics applications, in: Proceedings of the RSS Workshop: New Frontier for Deep Learning in Robotics, 2017.

Xiaofan Xu is a research engineer at Intel specializing in artificial intelligence and robotics. Previously, Xiaofan worked in the CTO office at Movidius on various research projects, including 3D volumetric object recognition using convolutional neural networks and training neural networks using synthetic data. Xiaofan holds a master degree in electronic engineering from Trinity College Dublin.



Sam Caulfield is a research engineer at Intel. His work mainly involves software development on low-power and memory-constrained devices. He graduated with a degree in computer science from Trinity College Dublin.

João Amaro (S14) received his B.Sc. degree in electrical and computer engineering from the Faculty of Science and Technology of the University of Coimbra (FCTUC), Portugal, where he also concluded the M.Sc. degree in computer architectures for ultrasound systems in 2013. During his B.Sc. and M.Sc. studies he was the recipient of three FCTUC 3% top student awards. His research interests span the development of new medical imaging reconstruction paradigms, parallel computing architectures, high-level synthesis, and efficient code portability. In 2015, João became a PhD student, researching advanced medical ultrasound imaging algorithms and heterogeneous computing architectures. In September 2016, João enrolled in a 3-month internship at Movidius, researching 3D volumetric object recognition using CNNs.

Gabriel Falcao (S07M10SM14) graduated in electrical and computer engineering from the University of Porto, where he also concluded the M.Sc. degree in digital signal processing. In 2010 he received the Ph.D. degree from the University of Coimbra, where he became an Assistant Professor. In 2011/2012 and again in 2017/18 Gabriel was a Visiting Professor at EPFL, Switzerland. In 2013 he was the recipient of a Google Faculty Research Award and the Altera Europe-Wide University contest 2012–2013. Presently, he is studying efficient parallelization strategies, novel algorithms and architectures for dealing with compute-intensive applications used in medical, imaging and deep neural network imaging contexts, in parallel with continuous work in digital communications. He is a researcher at Instituto de Telecomunicaes, and a senior Member of the IEEE, signal Processing society, and the HiPEAC Network of Excellence.

David Moloney is a director of Machine vision at Intel Corp. and has worked in the semiconductor industry since qualifying with a BEng from DCU in 1985. He has a wealth of international experience having worked for Infineon in Munich for 5 years and SGS- Thomson Microelectronics (STM) in Milan for 4 years respectively. In 1994 he returned from STM to lead the engineering team for the first product development at Parthus Technologies (https://www.ceva-dsp.com/) where he was a key member of the management team and where he spearheaded the development of the Parthus Bluetooth technology. David left Parthus in 2003 to work towards his PhD in Trinity College Dublin and as an independent consultant for Frontier Silicon and Dublin City University. He subsequently co-founded Movidius as CTO in 2005 which went on to pioneer low-power embedded vision and neural network processing in edge devices before being acquired by Intel in November 2016. He received a PhD from Trinity College Dublin in 2010 for his research into high performance computer architectures. His interests span SoC and embedded processor design, deep-learning hardware, computer vision, robotics and navigation. David (co-)inventor of 36 issued patents and (co-)author of 32 conference/journal papers (Publication List https://scholar.google.com/citations?user=WU3g6y8AAAAJ&hl=en).