Classsification of 3D Models

Lab Based Project

A project report submitted to Indian Institute of Technology Roorkee
in accordance with the requirements of the degree of

Bachelor of Technology

in

Computer Science and Engineering

Submitted by

Nitin Gaurav Singh (15118049)
Mahendra Kumar (15114043)
Anuj (15114015)
Satyendra Kumar Banjare (15114064)
Prashant Meena (14114038)

Under the guidance of

Dr. Sudip Roy

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY ROORKEE
ROORKEE - 247667 (INDIA)

April, 2018

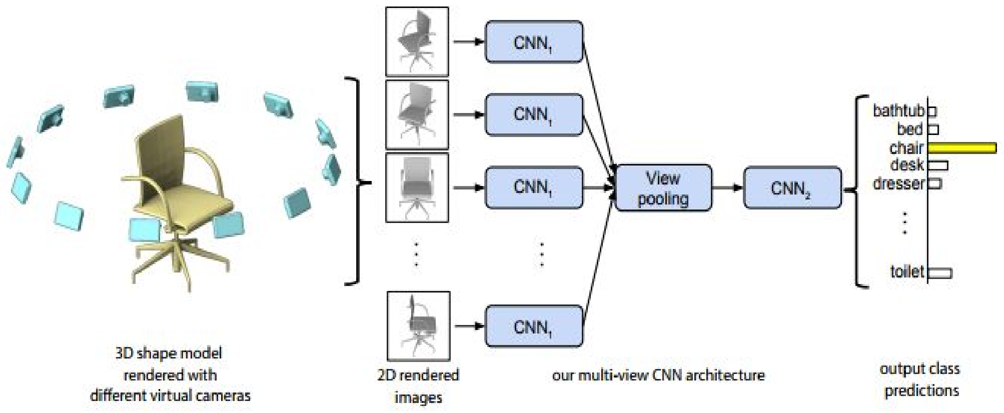
Introduction

3D shape models are becoming widely available and easier to capture, making available 3D information crucial for progress in object classification. Current state-of-the art methods rely on CNNs to address this problem.

So, in this project, we have implemented Volumetric CNNs for object classification on 3D data (binary volume). This work is based on *Volumetric and Multi-View CNNs for Object Classification on 3D Data* [2] .

Multi-view CNN - Our Initial Approach

This is an easy way to classify 3D object. Here we take images from different orientations of the object as shown in the below diagram. Then we apply CNN over these image to classify the model.



Neural network used:

Here we used an 8 layered architecture

[input]->1->2->3->4->5->6->[output].

All these layers are depicted in the following code.

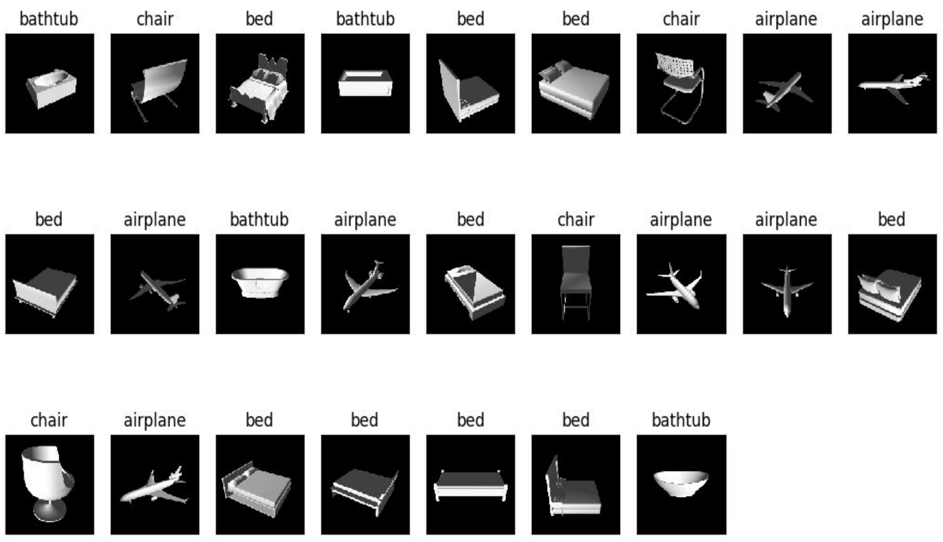
We used DNN (Deep Neural Network) for this.

Dataset used - ModelNet40 (https://shapenet.cs.stanford.edu/media/modelnet40_h5.tar)

The Neural Network Architecture for the Multi-View CNN Approach

```
tf.reset_default_graph()
convnet = input_data(shape=[None, IMG_SIZE, IMG_SIZE, 1], name='input')

convnet = conv_2d(convnet, 32, 5, activation='relu')
convnet = max_pool_2d(convnet, 5)

convnet = conv_2d(convnet, 64, 5, activation='relu')
convnet = max_pool_2d(convnet, 5)

convnet = conv_2d(convnet, 128, 5, activation='relu')
convnet = max_pool_2d(convnet, 5)

convnet = conv_2d(convnet, 64, 5, activation='relu')
convnet = max_pool_2d(convnet, 5)

convnet = conv_2d(convnet, 32, 5, activation='relu')
convnet = max_pool_2d(convnet, 5)

convnet = fully_connected(convnet, 1024, activation='relu')
convnet = dropout(convnet, 0.8)

convnet = fully_connected(convnet, 5, activation='softmax')
convnet = regression(convnet, optimizer='adam', learning_rate=LR,
loss='categorical_crossentropy', name='targets')

model = tflearn.DNN(convnet, tensorboard_dir='log')
```

Result Obtained:



Volumetric CNN and Feature Extraction - Our Final Approach

We have implemented this for the following different neural networks:

I. Auxiliary Training by Subvolume Supervision

We observe significant overfitting when we train the previous volumetric CNN methods. When the volumetric CNN overfits to the training data, it has no incentive to continue learning. We thus introduce auxiliary tasks that are closely correlated with the main task but are difficult to overfit, so that learning continues even if our main task is overfitted.

These auxiliary training tasks also predict the same object labels, but the predictions are made solely on a local subvolume of the input. Without complete knowledge of the object, the auxiliary tasks are more challenging, and can thus better exploit the discriminative power of local regions.

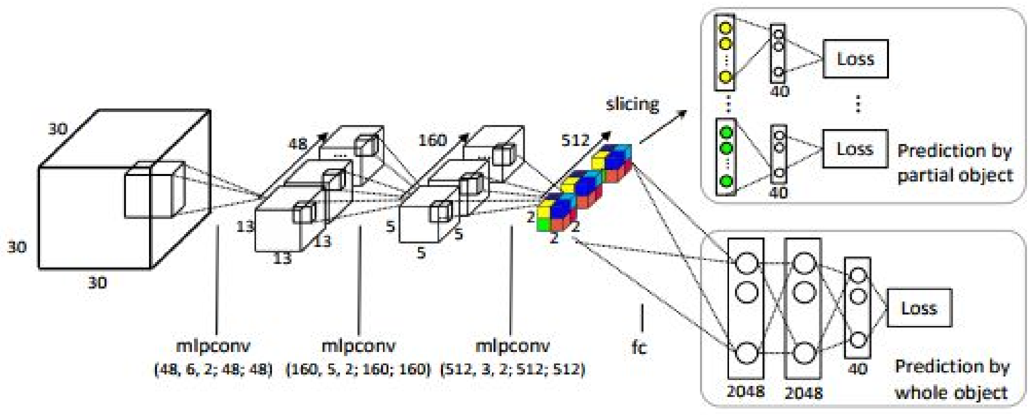


Fig- architecture of Auxiliary Training by Subvolume Supervision

The first three layers are mlpconv (multilayer perceptron convolution) layers. The input and output of our mlpconv layers are both 4D tensors. Compared with the standard combination of linear convolutional layers and max pooling layers, mlpconv has a three-layer structure and is thus a universal function approximator if enough neurons are provided in its intermediate layers. Therefore, mlpconv is a powerful filter for feature extraction of local patches, enhancing approximation of more abstract representations. In addition, mlpconv has been validated to be more discriminative with fewer parameters than ordinary convolution with pooling .

At the fourth layer, the network branches into two. The lower branch takes the whole object as input for traditional classification. The upper branch is a novel branch for auxiliary tasks. It slices the 512 × 2 × 2 × 2 4D tensor (2 grids along x, y, z axes and 512 channels) into 2×2×2 = 8 vectors of dimension 512. We set up a classification task for each vector.

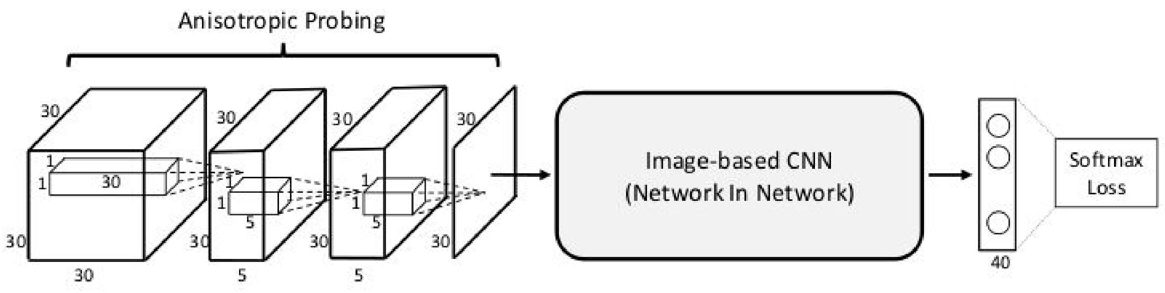
A fully connected layer and a softmax layer are then appended independently to each vector to construct classification losses. Simple calculation shows that the receptive field of each task is 22×22×22, covering roughly 2/3 of the entire volume.

II. Anisotropic Probing

Multi-view CNNs first project 3D objects to 2D and then make use of well-developed 2D image CNNs for classification.Inspired by its success, we design a neural network architecture that is also composed of the two stages. However, while multi-view CNNs use external rendering pipelines from computer graphics, we achieve the 3D-to-2D projection using network layers in a manner similar to 'X-ray scanning'.

Key to this network is the use of an elongated anisotropic kernel which helps capture the global structure of the 3D volume.The anisotropic probing module contains three convolutional layers of elongated kernels, each followed by a nonlinear ReLU layer. Note that both the input and output of each layer are 3D tensors.

The neural network has two modules: an anisotropic probing module and a network in network module.



Our anisotropic probing network is capable of capturing internal structures of objects through its X-ray like projection mechanism. This is an ability not offered by standard rendering. Combined with multi-orientation pooling (introduced below), it is possible for this probing mechanism to capture any 3D structure, due to its relationship with the Radon transform.In addition, this architecture is scalable to higher resolutions, since all its layers can be viewed as 2D. While 3D convolution involves computation at locations of cubic resolution, we maintain quadratic compute.

III. Multi-Orientation Pooling - future scope

The two networks proposed above are both sensitive to model orientation. Compared to Modelnet which only augments data by rotating around vertical axis, our experiment shows that orientation pooling combined with elevation rotation can greatly increase performance.

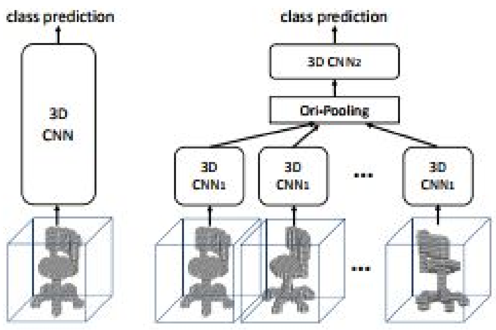


Fig- Multi-oriented pooling

Shape Descriptor

A large variety of shape descriptors has been developed in the computer vision and graphics community. For instance, shapes can be represented as histograms or bag-of-feature models which are constructed from surface normals and curvatures. Alternatives include models based on distances, angles, triangle areas, or tetrahedra volumes, local shape diameters measured at densely-sampled surface points, Heat kernel signatures, or extensions of SIFT and SURF feature descriptors to 3D voxel grids.

In contrast to recently developed feature learning techniques, these features are handcrafted and do not generalize well across different domains.

Project working

Installation

Install Torch7 (http://torch.ch/docs/getting-started.html).

Note that cuDNN and GPU are required for *VolumetricBatchNormalization* layer. You also need to install a few torch packages including cudnn.torch, cunn, hdf5 and xlua.

Usage

To train a model to classify 3D object:

```
th train.lua
```

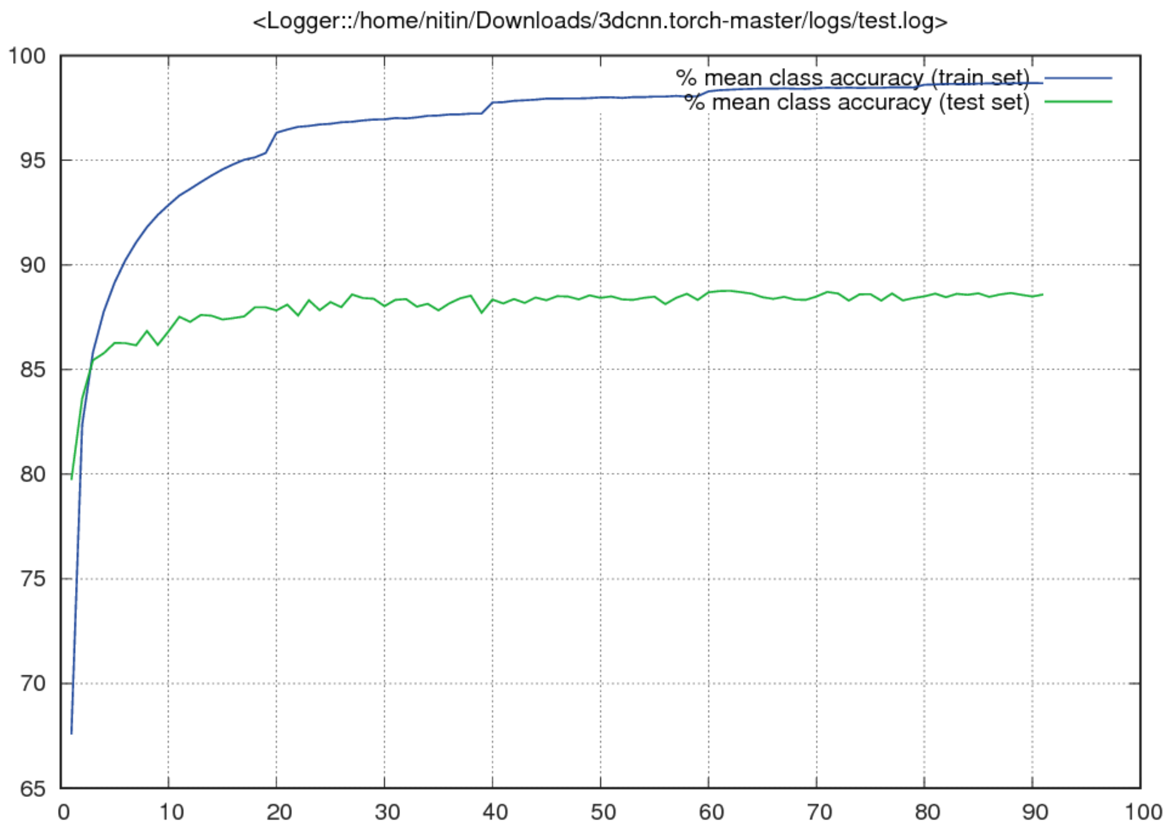
Voxelizations of ModelNet40 models in HDF5 files will be automatically downloaded (633MB).

Modelnet40_60x includes azimuth and elevation rotation augmented occupancy grids. Modelnet40_12x is of azimuth rotation augmentation only. Modelnet40_20x_stack is used for multi-orientation training. There are also text files in each folder specifying the sequence of CAD models in h5 files.

To see HELP for training script:

```
th train.lua -h
```

Results

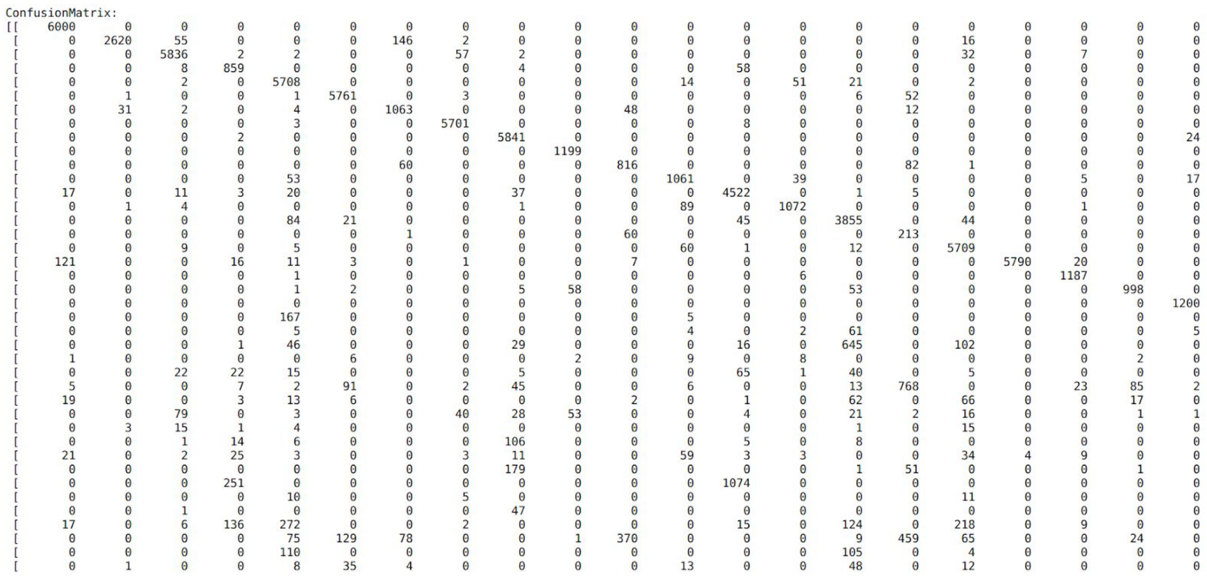


Learning Rate	0.000625
Evaluation Counter	838383
Weight Decay	0.0005
Learning Rate Decay	1e-07
Momentum	0.9

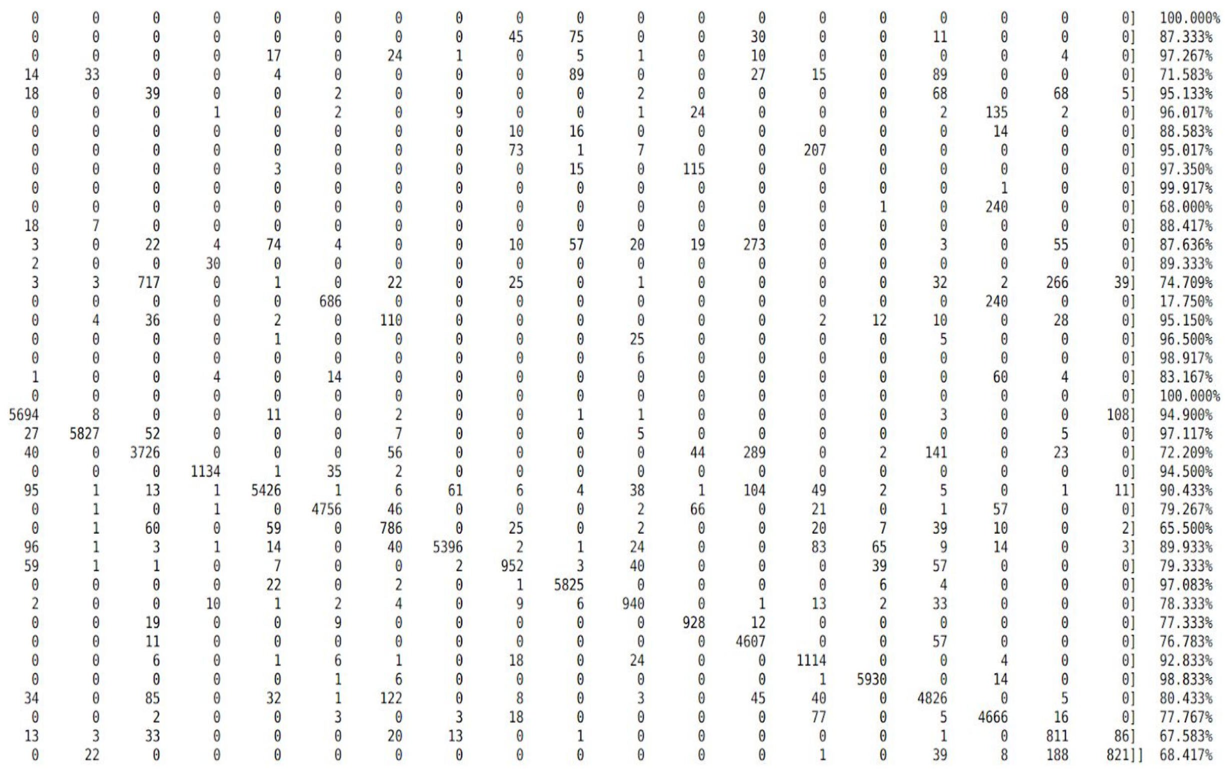
% mean class accuracy (train set) vs % mean class accuracy (test set)

Train set	Test set
6.76E+01	7.97E+01
8.24E+01	8.36E+01
8.58E+01	8.55E+01
8.78E+01	8.58E+01
8.92E+01	8.63E+01
9.02E+01	8.63E+01
9.11E+01	8.62E+01
9.29E+01	8.68E+01
9.48E+01	8.75E+01
9.63E+01	8.78E+01
9.72E+01	8.85E+01
9.79E+01	8.84E+01
9.80E+01	8.81E+01
9.84E+01	8.86E+01
9.85E+01	8.87E+01
9.85E+01	8.83E+01
9.87E+01	8.86E+01
9.87E+01	8.85E+01
9.87E+01	8.86E+01

Following is the Confusion Matrix obtained:



```
ConfusionMatrix:
[[   6000      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
 [      0   2620     55      0      0      0    146      2      0      0      0      0      0      0      0      0     16      0      0      0      0
 [      0      0   5836      2      2      0      0     57      2      0      0      0      0      0      0      0     32      0      7      0      0
 [      0      0      8    859      0      0      0      0      4      0      0      0     58      0      0      0      0      0      0      0      0
 [      0      0      2      0   5708      0      0      0      0      0      0     14      0     51     21      0      2      0      0      0      0
 [      0      1      0      0      1   5761      0      3      0      0      0      0      0      0      6     52      0      0      0      0      0
 [      0     31      2      0      4      0   1063      0      0      0     48      0      0      0      0     12      0      0      0      0      0
 [      0      0      0      0      3      0      0   5701      0      0      0      0      8      0      0      0      0      0      0      0      0
 [      0      0      0      2      0      0      0      0   5841      0      0      0      0      0      0      0      0      0      0      0     24
 [      0      0      0      0      0      0      0      0      0   1199      0      0      0      0      0      0      0      0      0      0      0
 [      0      0      0      0      0      0     60      0      0      0    816      0      0      0      0     82      1      0      0      0      0
 [      0      0      0      0     53      0      0      0      0      0      0   1061      0     39      0      0      0      0      5      0     17
 [     17      0     11      3     20      0      0      0     37      0      0      0   4522      0      1      5      0      0      0      0      0
 [      0      1      4      0      0      0      0      0      1      0      0     89      0   1072      0      0      0      0      1      0      0
 [      0      0      0      0     84     21      0      0      0      0      0      0     45      0   3855      0     44      0      0      0      0
 [      0      0      0      0      0      0      1      0      0      0     60      0      0      0      0    213      0      0      0      0      0
 [      0      0      9      0      5      0      0      0      0      0      0     60      1      0     12      0   5709      0      0      0      0
 [    121      0      0     16     11      3      0      1      0      0      7      0      0      0      0      0      0   5790     20      0      0
 [      0      0      0      0      1      0      0      0      0      0      0      0      0      6      0      0      0      0   1187      0      0
 [      0      0      0      0      1      2      0      0      5     58      0      0      0      0     53      0      0      0      0    998      0
 [      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0   1200
 [      0      0      0      0    167      0      0      0      0      0      0      5      0      0      0      0      0      0      0      0      0
 [      0      0      0      0      5      0      0      0      0      0      0      4      0      2     61      0      0      0      0      0      5
 [      0      0      0      1     46      0      0      0     29      0      0      0     16      0    645      0    102      0      0      0      0
 [      1      0      0      0      0      6      0      0      0      2      0      9      0      8      0      0      0      0      0      2      0
 [      0      0     22     22     15      0      0      0      5      0      0      0     65      1     40      0      5      0      0      0      0
 [      5      0      0      7      2     91      0      2     45      0      0      6      0      0     13    768      0      0     23     85      2
 [     19      0      0      3     13      6      0      0      0      0      2      0      1      0     62      0     66      0      0     17      0
 [      0      0     79      0      3      0      0     40     28     53      0      0      4      0     21      2     16      0      0      1      1
 [      0      3     15      1      4      0      0      0      0      0      0      0      0      0      1      0     15      0      0      0      0
 [      0      0      1     14      6      0      0      0    106      0      0      0      5      0      8      0      0      0      0      0      0
 [     21      0      2     25      3      0      0      3     11      0      0     59      3      3      0      0     34      4      9      0      0
 [      0      0      0      0      0      0      0      0    179      0      0      0      0      0      1     51      0      0      0      1      0
 [      0      0      0    251      0      0      0      0      0      0      0      0   1074      0      0      0      0      0      0      0      0
 [      0      0      0      0     10      0      0      5      0      0      0      0      0      0      0      0     11      0      0      0      0
 [      0      0      1      0      0      0      0      0     47      0      0      0      0      0      0      0      0      0      0      0      0
 [     17      0      6    136    272      0      0      2      0      0      0      0     15      0    124      0    218      0      9      0      0
 [      0      0      0      0     75    129     78      0      0      1    370      0      0      0      9    459     65      0      0     24      0
 [      0      0      0      0    110      0      0      0      0      0      0      0      0      0    105      0      4      0      0      0      0
 [      0      1      0      0      8     35      4      0      0      0      0     13      0      0     48      0     12      0      0      0      0
```



```
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0]  100.000%
     0     0     0     0     0     0     0     0    45    75     0     0    30     0     0    11     0     0     0]  87.333%
     0     0     0     0    17     0    24     1     0     5     1     0    10     0     0     0     0     4     0]  97.267%
    14    33     0     0     4     0     0     0     0    89     0     0    27    15     0    89     0     0     0]  71.583%
    18     0    39     0     0     2     0     0     0     0     2     0     0     0     0    68     0    68     5]  95.133%
     0     0     0     1     0     2     0     9     0     0     1    24     0     0     0     2   135     2     0]  96.017%
     0     0     0     0     0     0     0     0    10    16     0     0     0     0     0     0    14     0     0]  88.583%
     0     0     0     0     0     0     0     0    73     1     7     0     0   207     0     0     0     0     0]  95.017%
     0     0     0     0     3     0     0     0     0    15     0   115     0     0     0     0     0     0     0]  97.350%
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     1     0     0]  99.917%
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     1     0   240     0     0]  68.000%
    18     7     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0]  88.417%
     3     0    22     4    74     4     0     0    10    57    20    19   273     0     0     3     0    55     0]  87.636%
     2     0     0    30     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0]  89.333%
     3     3   717     0     1     0    22     0    25     0     1     0     0     0     0    32     2   266    39]  74.709%
     0     0     0     0     0   686     0     0     0     0     0     0     0     0     0     0   240     0     0]  17.750%
     0     4    36     0     2     0   110     0     0     0     0     0     0     2    12    10     0    28     0]  95.150%
     0     0     0     0     1     0     0     0     0     0    25     0     0     0     0     5     0     0     0]  96.500%
     0     0     0     0     0     0     0     0     0     0     6     0     0     0     0     0     0     0     0]  98.917%
     1     0     0     4     0    14     0     0     0     0     0     0     0     0     0     0    60     4     0]  83.167%
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0]  100.000%
  5694     8     0     0    11     0     2     0     0     1     1     0     0     0     0     3     0     0   108]  94.900%
    27  5827    52     0     0     0     7     0     0     0     5     0     0     0     0     0     0     5     0]  97.117%
    40     0  3726     0     0     0    56     0     0     0     0    44   289     0     2   141     0    23     0]  72.209%
     0     0     0  1134     1    35     2     0     0     0     0     0     0     0     0     0     0     0     0]  94.500%
    95     1    13     1  5426     1     6    61     6     4    38     1   104    49     2     5     0     1    11]  90.433%
     0     1     0     1     0  4756    46     0     0     0     2    66     0    21     0     1    57     0     0]  79.267%
     0     1    60     0    59     0   786     0    25     0     2     0     0    20     7    39    10     0     2]  65.500%
    96     1     3     1    14     0    40  5396     2     1    24     0     0    83    65     9    14     0     3]  89.933%
    59     1     1     0     7     0     0     2   952     3    40     0     0     0    39    57     0     0     0]  79.333%
     0     0     0     0    22     0     2     0     1  5825     0     0     0     0     6     4     0     0     0]  97.083%
     2     0     0    10     1     2     4     0     9     6   940     0     1    13     2    33     0     0     0]  78.333%
     0     0    19     0     0     9     0     0     0     0     0   928    12     0     0     0     0     0     0]  77.333%
     0     0    11     0     0     0     0     0     0     0     0     0  4607     0     0    57     0     0     0]  76.783%
     0     0     6     0     1     6     1     0    18     0    24     0     0  1114     0     0     4     0     0]  92.833%
     0     0     0     0     0     1     6     0     0     0     0     0     0     1  5930     0    14     0     0]  98.833%
    34     0    85     0    32     1   122     0     8     0     3     0    45    40     0  4826     0     5     0]  80.433%
     0     0     2     0     0     3     0     3    18     0     0     0     0    77     0     5  4666    16     0]  77.767%
    13     3    33     0     0     0    20    13     0     1     0     0     0     0     0     1     0   811    86]  67.583%
     0    22     0     0     0     0     0     0     0     0     0     0     0     1     0    39     8   188   821]]  68.417%
```

CONFUSION MATRIX:

A confusion matrix (shown above) is a summary of prediction results on a classification problem.

The number of correct and incorrect predictions are summarized with count values and broken down by each class. This is the key to the confusion matrix.

The confusion matrix shows the ways in which your classification model is confused when it makes predictions.

It gives you insight not only into the errors being made by your classifier but more importantly the types of errors that are being made. It is this breakdown that overcomes the limitation of using classification accuracy alone.

The Architecture of the Neural Network used

```
nn.Sequential {
  [input -> (1) -> (2) -> (3) -> (4) -> (5) -> (6) -> (7) -> (8) -> (9)
-> (10) -> (11) -> (12) -> (13) -> (14) -> (15) -> (16) -> (17) -> (18)
-> (19) -> (20) -> (21) -> (22) -> (23) -> (24) -> (25) -> (26) -> (27)
-> (28) -> (29) -> (30) -> (31) -> (32) -> (33) -> (34) -> (35) ->
output]
  (1): cudnn.VolumetricConvolution(1 -> 48, 6x6x6, 2,2,2)
  (2): cudnn.VolumetricBatchNormalization
  (3): cudnn.ReLU
  (4): cudnn.VolumetricConvolution(48 -> 48, 1x1x1)
  (5): cudnn.VolumetricBatchNormalization
  (6): cudnn.ReLU
  (7): cudnn.VolumetricConvolution(48 -> 48, 1x1x1)
  (8): cudnn.VolumetricBatchNormalization
  (9): cudnn.ReLU
  (10): nn.Dropout(0.200000)
  (11): cudnn.VolumetricConvolution(48 -> 96, 5x5x5, 2,2,2)
  (12): cudnn.VolumetricBatchNormalization
  (13): cudnn.ReLU
  (14): cudnn.VolumetricConvolution(96 -> 96, 1x1x1)
  (15): cudnn.VolumetricBatchNormalization
  (16): cudnn.ReLU
  (17): cudnn.VolumetricConvolution(96 -> 96, 1x1x1)
  (18): cudnn.VolumetricBatchNormalization
  (19): cudnn.ReLU
  (20): nn.Dropout(0.200000)
  (21): cudnn.VolumetricConvolution(96 -> 512, 3x3x3, 2,2,2)
  (22): cudnn.VolumetricBatchNormalization
  (23): cudnn.ReLU
  (24): cudnn.VolumetricConvolution(512 -> 512, 1x1x1)
  (25): cudnn.VolumetricBatchNormalization
  (26): cudnn.ReLU
  (27): cudnn.VolumetricConvolution(512 -> 512, 1x1x1)
  (28): cudnn.VolumetricBatchNormalization
  (29): cudnn.ReLU
  (30): nn.Dropout(0.200000)
  (31): nn.View(4096)
  (32): nn.Linear(4096 -> 512)
  (33): cudnn.ReLU
  (34): nn.Dropout(0.500000)
  (35): nn.Linear(512 -> 40)
}
```

Model	Average Instance accuracy	Average Category accuracy
Subvol supervision	0.89262560778	0.860511627907
Anisotropic probing	0.899513776337	0.860720930233

We have included batch normalization and drop layers after all the convolutional layers (in caffe prototxt there are only dropouts after fc layers and no batch normalization is used).

Besides the two models proposed in the paper (subvolume supervised network and anisotropic probing network), we have also found a variation of the base network used in subvolume_sup, which we call 3dnin_fc here. 3dnin_fc has a relatively simple architecture (3 mlpconv layers + 2 fc layers) and performs also very well, so we have set it as the default architecture to use in this repository.

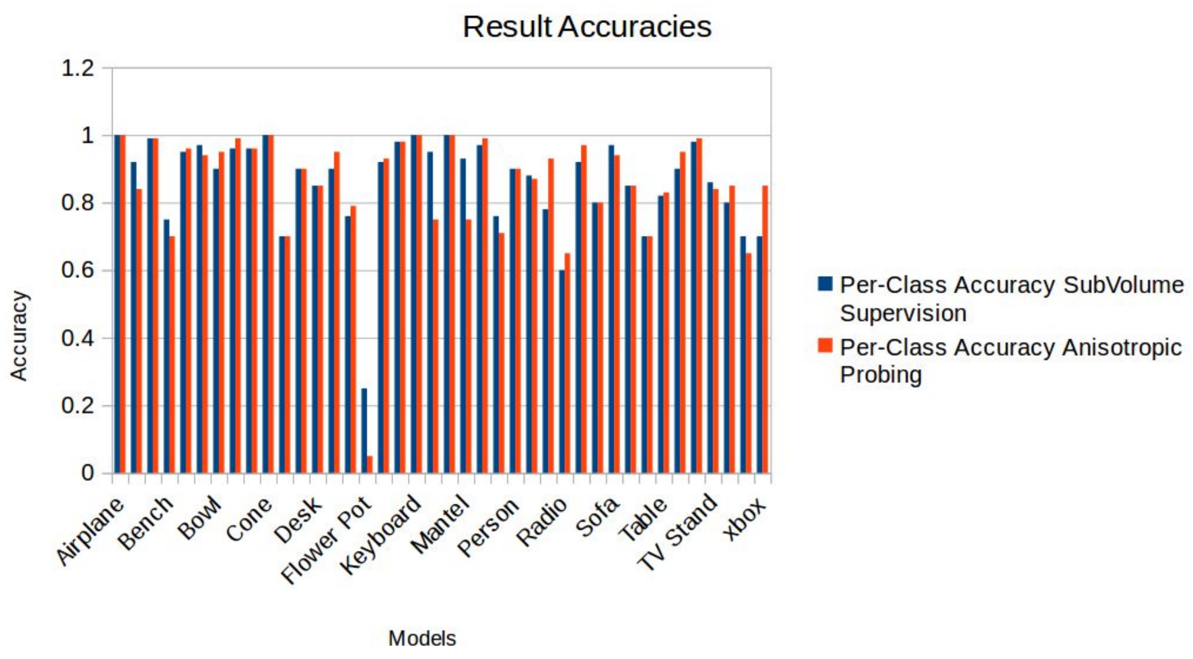
Numbers reported in the table above are average instance accuracy on the whole ModelNet40 test set containing 2468 CAD models from 40 categories.This different from what is on the modelnet website, which is average class accuracy on either a subset of the test set or the full test set.

Caffe Models and Reference Results

Caffe prototxt files of models reported in the paper have been included in the caffe_models directory.

Class	*No. Of Shapes*	*Per-Class Accuracy SubVolume Supervision*	*Per-Class Accuracy Anisotropic Probing*
Airplane	100	1.00	1.00
Bathtub	50	0.92	0.84
Bed	100	0.99	0.99
Bench	20	0.75	0.70
Bookshelf	100	0.95	0.96
Bottle	100	0.97	0.94
Bowl	20	0.90	0.95
Car	100	0.96	0.99
Chair	100	0.96	0.96
Cone	20	1.00	1.00
Cup	20	0.70	0.70
Curtain	20	0.90	0.90
Desk	86	0.85	0.85
Door	20	0.90	0.95
Dresser	86	0.76	0.79
Flower Pot	20	0.25	0.05
Glass Box	100	0.92	0.93
Guitar	100	0.98	0.98
Keyboard	20	1.00	1.00
Lamp	20	0.95	0.75

Laptop	20	1.00	1.00
Mantel	100	0.93	0.75
Monitor	100	0.97	0.99
Night Stand	86	0.76	0.71
Person	20	0.90	0.90
Piano	100	0.88	0.87
Plant	100	0.78	0.93
Radio	20	0.60	0.65
Range Hood	100	0.92	0.97
Sink	20	0.80	0.80
Sofa	100	0.97	0.94
Stairs	20	0.85	0.85
Stool	20	0.70	0.70
Table	100	0.82	0.83
Tent	20	0.90	0.95
Toilet	100	0.98	0.99
TV Stand	100	0.86	0.84
Vase	100	0.80	0.85
Wardrobe	20	0.70	0.65
xbox	20	0.70	0.85



References

1. http://vis-www.cs.umass.edu/mvcnn/docs/su15mvcnn.pdf [Multi-view Convolutional Neural Networks for 3D Shape Recognition]
2. https://arxiv.org/pdf/1604.03265.pdf [Volumetric and Multi-View CNNs for Object Classification on 3D Data]
3. http://cs231n.stanford.edu/reports/2016/pdfs/417_Report.pdf [Neural Network for 3D object classification]
4. https://machinelearningmastery.com/confusion-matrix-machine-learning/ [Confusion matrix]
5. http://www.cs.princeton.edu/~funk/smi01.pdf [Matching 3D Models with Shape Distributions]
6. http://cs231n.github.io/convolutional-networks/ [Neural Networks for visual recognition]
7. https://shapenet.cs.stanford.edu/media/modelnet40_h5.tar [Dataset ModelNet40]