Classification of 3D Shapes with Convolutional Neural Networks

Leonid Keselman
Stanford University
leonidk@stanford.edu

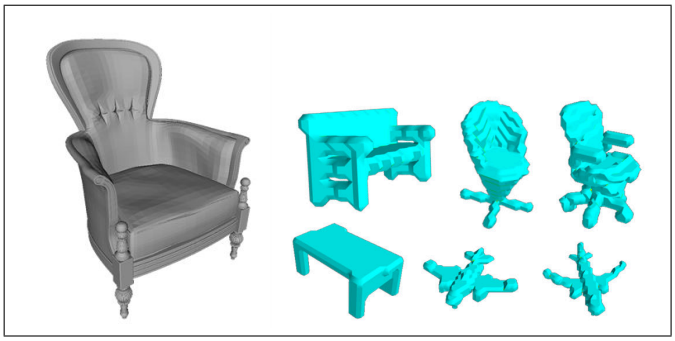

Figure 1. A visual example of the data used in this report. On the left is a mesh from the ShapeNet dataset (from the 'chair' class), while the examples on the right are 30 x 30 x 30 voxel grids. Examples of the 'desk', 'bench', 'airplane' and 'chair' classes are provided. Only the surface voxels are visible and the blue color is for visualization, as the data itself is simply binary.

Abstract

This project explores how to apply convolutional neural networks to efficiently classify 3D shapes. Using 3D shapes from the ShapeNet project [1], they are first centered, rescaled and resampled to occupancy voxel grids of dimension 30 x 30 x 30. We implement 3D convolutional neural networks operating directly on these three dimensional volumes, as well as 2D convolutional networks operating on a learned embedded from the 3D model. Results from both methods are state-of-the-art compared to existing, published methods.

1. Introduction

Recently, with the the ImageNet challenge in image classification [15], many other large scale datasets have been collected for the purpose of creating competitive, quantitative progress in machine learning research. Specifically, this project focuses on ShapeNet [1], a recently released dataset that includes 3D CAD models of forty different object categories, dubbed ModelNet40. This paper presents various designs and tests of convolutional neural networks to solve this object classification challenge. In this case, we will present both 2D and 3D convolutional networks, both trained from scratch and refined on existing datasets.

1.1. Dataset format

In this paper, we will train and test on the ModelNet40 dataset released from the ShapeNet dataset [1, 22]. The dataset is already split into a test set and a training set. In my own use, the training set is further split into a training set and a validation set, and the validation set is used for hyper-parameter tuning, although the reported results in the paper are test set accuracies. The authors of the original paper have kept track of published results on this dataset and their table is replicated in full, with citations as Table 1 on page 7.

Unlike images, in which there is a single standard representation, namely pixel grids of luminosity, shapes have a wide range of representations, many of them incompatible in nature [12]. In fact, shapes represent a very different type of object than an image. While images represent the projection of a three-dimensional scene (four if you count motion blur), shapes represent an R^2 surface embedded in R^3. Traditionally, shapes are stored as meshes, which are a list of triangular or quadrilateral components, unordered and with no connectivity information. This is similar to using vector shapes to represent 2D content, and has the downside of providing minimal locality and connectivity information. A straightforward extension of the pixel model of images (that is, a quantized representation as a rank-2 tensor) to shapes is to represent them as a rank-3 tensor, or volume. While many volume models exist, we chose to use occupancy grids, which are represent the inside of a shape with ones and all other elements as zero. An alternative volume representation that we wish to explore in future work is that of level sets or signed distance fields [12], which are more information dense, but are less common in 2D. Our volume size is 30 x 30 x 30, which was used in the original paper and will be replicated for consistent results [1]. However, as seen in figure 1, such volumes are fairly low resolution, but still require the same memory footprint as an 165x165

image. As an implementation detail, the datasets were converted and stored as shuffled HDF5 files. The dataset contains 12,311 files, with a roughly 70%/30% training and test split already provided. A smaller subset of the training set was used for validation, but all the results reported here are on the standard test set provided.

2. Related Work

The original ShapeNets paper, which introduced the ModelNet40 dataset [22], implemented a 3D volumetric convolution approach to classify the dataset. They report 77% top one accuracy, which suggests that the dataset is fairly easy for convolutional neural networks to fit, as top one accuracy performance on ImageNet is still below 30% [5]. Of interest, they report pre-training the data with a deep-belief-net architecture and contrastive divergence [6]. They also report that using trained neural networks significantly reports the hand-designed features used in the previous literature, namely Light Field descriptors and Spherical Harmonic descriptors [2].

The other published work using 3D convolutions is VoxNet [13], which uses a more traditional CNN architecture, with only gradient-based training methods. This mirrors those used in seminal works in digit [10] and image classification [9]. Our implemented 3D CNN work draws on a similar architecture, design, and training procedure. Their work was sufficient to bring state-of-the-art to 83.0% top one accuracy.

However, there has been another line of work on this dataset, that focusing on using two dimensional convolutional networks. This circumvents the intense memory requirements of 3D CNNs, where data is typically in R^5 (width, length, height, filter number, minibatch size). The first work, DeepPano [17], used a cylindrical projection of the mesh, and only fed the network a single view. This circumvented the need to select a viewpoint.The results, obtaining a 77.6% accuracy, were an improvement on the originally published ShapeNet paper.

Most recently, at ICCV 2015, Multiview CNNs were published, which used a set of sampled projections of the mesh. Their results only used 2D convolutions, were pretrained using ImageNet based classifiers, and were capable of accepting multiple viewpoints. Their best performance with a single viewpoint, of 85.1%, was a significant improvement on both the 3D CNNs and single view 2D CNNs. As an even more impressive result, their performance with 80 views was 90.1%.

Based on this impressive performance, there may be evidence to believe that 2D projections of 3D shapes, classified with 2D CNNs, may perform better than using straightforward 3D CNNs. While the authors have some thoughts on this matter, perhaps connecting the locality of CNN models with how projections create a small-world network [21] across the R^2 surface of a shape, this theory is neither tested nor elaborated upon in this project.

However, it seems that a natural question to ask is, can we come up with a way to learn a robust 3D to 2D projection? That is the motivating example behind most of the methods implemented below, which seem to implement an embedding network to transform 30 x 30 x 30 volumes directly into an image sized dimension for shape classification. The goal is that this non-linear transformation will arise from gradient learning methods.

A table summarized the performance of all previously published methods and all methods implemented herein is available as Table 1 on page 7.

3. Methods

In general, this work uses convolutional neural networks, with pooling, convolution, and fully connected layers [10]. For this work there are two general classes of model implemented, 3D CNNs such as those implemented in [13], and 2D CNNs such as those implemented in [18], with the caveat that a 3D to 2D embedded is learned rather than developed using a heuristic. An appropriate section is available to describe each class of methods, but they both have several things in common. Namely, all models are learned using a ReLU [14] nonlinearlity, convolutional layers include L_2 regularization, and the model is a loss function is a softmax training loss, which implements a cross-entropy loss. Formally,

$$Loss = \frac{1}{N}\sum_i L_i + \lambda R(W)$$

$$L_i = -\log(\frac{e^{f_{y_i}}}{\sum_j e^{f_i}})$$

$$R(W) = \sum_k \sum_l W_{k,l}^2$$

This should provide a more unique solution across the space of possible weights, while also preferring a high probability value for the correct answer and a low probability for all incorrect answers. Additionally, all models were initialized with mostly Xavier [4] initialization, and slightly positive biases, unless otherwise noted. Models were all trained directly with gradient-based learning, using backpropagation and mini-batch gradient descent. Batches were typically sized to fit in memory and gradient were accumulated until batches were at least size 32. Learning rates (α) were set to 0.001, and dropped by a power of ten every 2 epochs. Typically convergence is quick with 5-8 epochs required. The update rule was typically SGD with momentum (γ set to 0.9), as follows, although some alternatives were mentioned below.

$$v = \gamma v + \alpha \nabla Loss$$

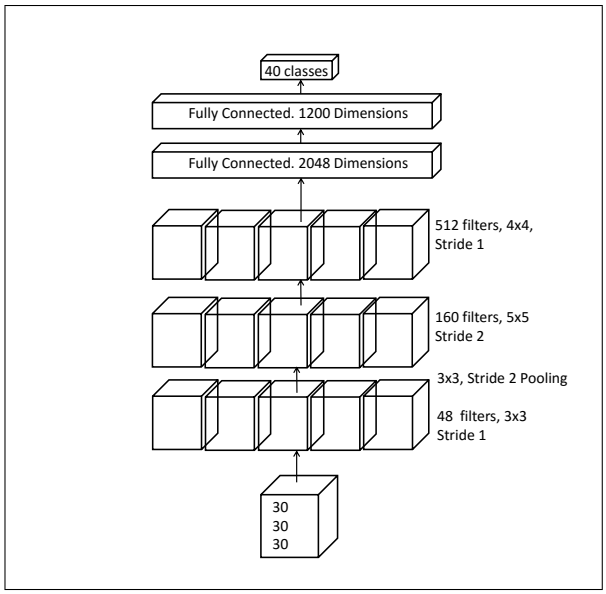

Figure 2. A view of the architecture used for the 3D CNN. This shows the most complete and best performing architecture, although others were tried, with larger convolutional filters, smaller fully connected layers, and without pooling. Some selected results are reported in Table 1. Additionally, dropout of 50% probability was used before both fully connected layers.

$$W = W - v$$

In general, most of this work will be done using existing CNN (convolutional neural network) packages. I've used Torch [3] for my 3D CNN networks, and Caffe [7] for my fine-tuned embedded networks. This comes from Caffe's convenience for tweaking layerwise learning rates and fine-tuning from the Model Zoo. On the other hand, Torch has better support for volumeric operations (namely, Torch has VolumetricPooling while Caffe did not).

3.1. 3D Convolutional Neural Network

As seen in figure 2, a fairly straightforward CNN architecture, with three convolutional layers and two fully-connected layers was used to train a network. While two convolutional layer models have been published before [13], we decided to follow the recent trend of going deeper [19] and build a three convolutional layer model; we also used a traditional two layer fully connected network on the top [10] After some experimentation with different solvers (SGD, Adam [8], RMSProp[20]), we were able to get very good performance, exceeding the previously published results for 3D CNNs. Using computing resources from the Graphics Lab at Stanford, it takes about hour to train a model to convergence on a single Tesla K40. These models were not heavily tuned or optimized, as their goal was primarily to serve a baseline for the 2D embedding models below. The tuning of hyperparameters received help and

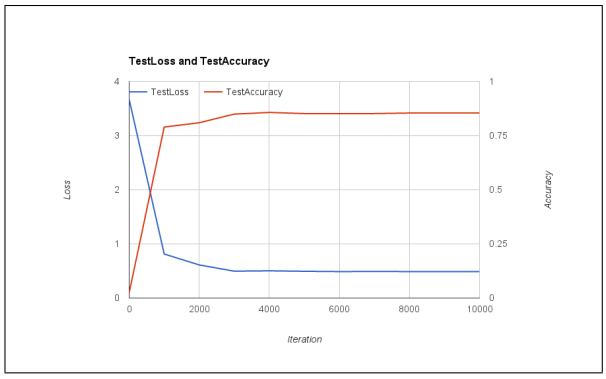

Figure 3. An example of our Upconv2 network being trained directly from VGG-16 weights. The plot shows test accuracy and loss, but this was simply because the graph was generated from model snapshots. Actual results during design were only on the validation set

consultation provided by Charles Qi and Hao Su from Professor Guibas' Geometric Computation group. Some interesting results were that Adam converged very quickly to a good result in a single epoch, but saturated in performance compared to vanilla SGD, which took longer to converge but did so to a better result. Additionally, dropout provided an improvement of a few percentage points, and was applied at the fully connected layers only. Additionally, a small amount of max pooling was beneficial after the bottom convolution layer. The best model generated obtained 86.7% top one accuracy on the ModelNet40 dataset.

3.2. Learning a 3D to 2D embedding

Since the current best performer on ModelNet40 uses pre-trained CNNs on multi-view renders of ModelNet, it seems natural to propose that we use a gradient-trainable transformation to directly feed ModelNet data to a pre-trained image network, with no intermediate steps in between. There are multiple advantages of those approach. First, the memory hit of using volumetric convolution (which is typically N^5 with minibatches, width, height, length, and filter bank size) is avoided. Second, this allows us to learn a transformation that may be better than the heuristic transformations used in existing 2D-based shape classifiers. Third, we can exploit ImageNet's pre-training as a regularizer, or prior, so that the trained embedding layer will create images that follow natural image statistics and don't overfit our small dataset as quickly. Additionally, systems which deploy both ShapeNet and ImageNet classifiers can reuse weights and a computational routine between them, allowing for an increased value and use of accelerating existing ImageNet classifiers.

The basic idea is straightforward, learn an approach to take 30 x 30 x 30 volumetric data and find a way to output 224 x 224 x 3 image data. This embedding network is then

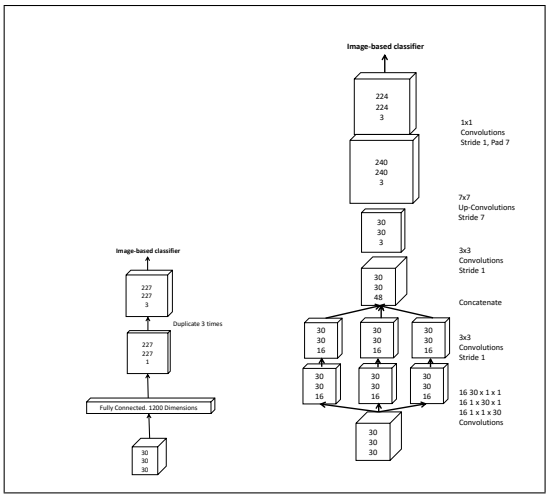

Figure 4. A view of the various architectures for 3D to 2D Embedding networks. On the left is a basic fully connected embedding with just a single matrix multiply. To reduce the parameter space, this network is only done on a single channel and then duplicated thrice for a three channel input image. The number of parameters for such a network is easily 80 million, even for the single channel version. To the right is an upsampling-based architecture, which has far fewer parameters and generates more natural looking images; this is our Upconv1 architecture

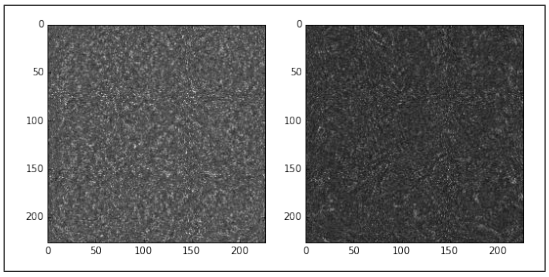

Figure 5. Two example images generated by using a fully connected network. The images look like complete noise, but still achieve 75% top one accuracy in classification performance by using CaffeNet with fine-tuned fc7 features [7]. The images are normalized for 0 to 1 for display.

used to generate images which can be fed into a normal ImageNet network. The network can then either be trained from scratch, fine-tuned entirely, or fine-tuned for just the top layer features. There are multiple possible architectures for such an embedding and three at explored in detail here. The two basic differences are summarized in figure 4. All these models converge fairly smoothly and quickly and an example of their performance is shown in figure 3. The models were all trained on Amazon's AWS EC2 instances with Nvidia K520 GPUs.

3.2.1 Learning a full-connected embedding

The primary challenge of connecting an ImageNet network to ModelNet is that the dimensions of the data are very different, and the naive solution is the use of a fully-connected layer. As seen in figure 4, it's fairly staightforward to use a traditional, single layer fully connected layer (with a nonlinearility) to perform embedding. However, this creates about 80 million parameters, even by doing this over greyscale images and duplicating the image across all channels. Our initial experiments fine-tuned a pre-trained CaffeNet [7], with a fully-connected layer hooking up the ModelNet input to an ImageNet sized image. The challenge here is that the representation is very low dimensional (1024), the number of parameters is very large, and the images look like noise. We were able to tune these networks to begin training and converge, but their performance was significantly worse than that of 3D CNNs.

To further reduce the number of parameters, we moved to fine-tuning a Network-in-Network [11] architecture, which has no fully-connected layers at the top, and instead uses global average pooling and $1x1$ convolutions to reduce down to a classifier. We were also able to modify and train this network to converge, although it took longer and performed worse than the CaffeNet model. These fine-tuned networks were trained on a CPU, specifically, a four core desktop computer in Caffe (on Windows!) and took roughly 3-4 hours to converge. GoogleNet was also fine-tuned, and also performed poorly.

To enhance the performance of the FC layer, we locked the CaffeNet weights, and only fine-tuned the FC7 features and the embedding network, along with trying 3 1x1 convolutions on the volume (one down each cardinal axis of the volume, equivalent to 1x1x30, 1x30x1,30x1x1 convolutions), to generate a 30x30x3 volume, which was then reduced to size 30x30x1 and learned fully connected with the single channel image size. While this slightly reduced the dimensionality of the vector (900 dimensions), the more structured learning allowed for a large increase in performance (up to 75%). An example of these embeddings can be seen in 5.

3.2.2 Learning a convolutional embedding

In order to reduce the burden of a fully-connected layer, we began to explore a more convolutional architecture to creating an embedding between the 3D ModelNet and 2D ImageNet data. The basic building block was a deconvolution, or an upconvolution layer as we prefer to call it, where a single input pixel is mapped to multiple output pixels. These were recently introduced in the scope of full-frame segmentation [23], and are being deployed here to avoid the high dimensional fully connected layer.

One challenge to moving to these upconvolution layers

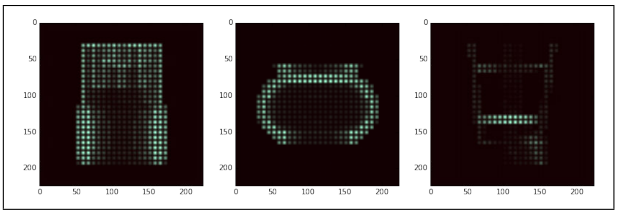

Figure 6. Upconv1 results. These examples are all predicted correctly and are of classes (left-to-right) of 'cup', 'bowl', 'desk'. There are visible holes from using non-overlapping convolutions for our upsampling, and these are fixed by our upconv2 architecture. The images are normalized for 0 to 1 for display.

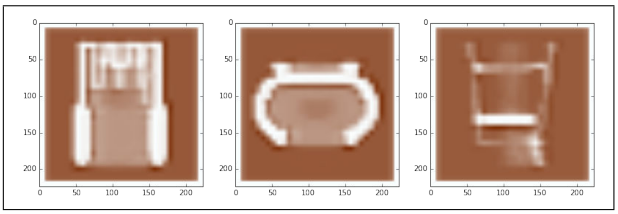

Figure 7. Upconv2 examples. These examples are all predicted correctly and are of classes (left-to-right) of 'cup', 'bowl', 'desk'. The images are normalized for 0 to 1 for display.

is that the dimensions of our data is even, which requires an even sized output image layer if we want symmetric edge performance in between. To achieve this, we only implement these upconvolution models on 224 sized ImageNet networks, such as ResNet[5] and VGG [16], but not CaffeNet, which is sized 227x227.

Our basic design idea (as seen in figure 4) was to continue with the 1x1 convolution design we used in the fully connected case, where 1x1x30, 1x30x1 and 30x1x1 convolutions are used down each cardinal axis. They're then transformed with another convolution layer, concatenated, and passed through upsampling layers. These then have to be cropped with a 1x1 convolution to ImageNet size. The upconvolution layer has to be carefully initialized with bilinear weights, as using Xavier initialization leads to a failure of the optimization to converge in our testing. An example of these embeddings can be seen in 6.

Our initial results with the upconvolution layer in were very successful, but we noticed some issues with holes in the data due to our choice of non-overlapping convolution kernels. We thus implemented 8 to alleviate these issues. It is denoted as Upconv2 in our results, and some example embeddings can be seen in 7.

3.2.3 Picking an ImageNet network

We tried our embedding results across many various ImageNet architectures. In general, we found VGG-16 to perform best in our testing. ResNet and GoogleNet report better ImageNet performance but were difficult to train in our testing.

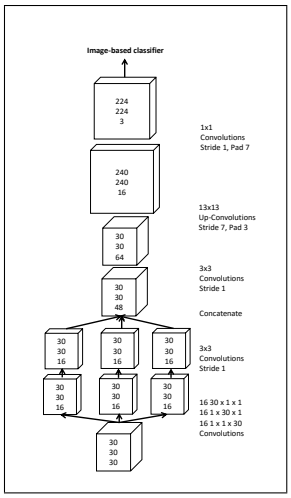

Figure 8. An overview of our improved upconvolution architecture: Upconv2.

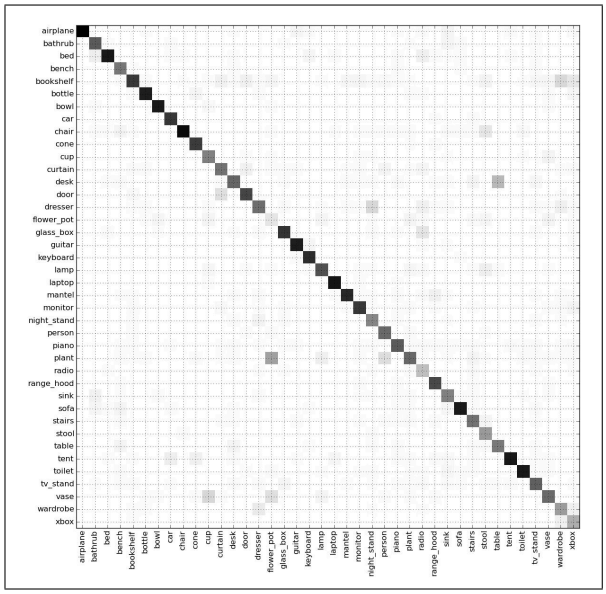

Figure 9. Confusion matrix for a 1x1 + Locked CaffeNet style method (2D convolutions)

4. Results

The summary table of results can be seen in table 1. The previously published methods are at the top with citations numbers. The 3D CNN networks, as described in figure 2, built in Torch and trained from scratch (on a GPU) are listed in the middle section. While results from three different embedding architectures, as shown in figures 4 and 8 are seen at the end.

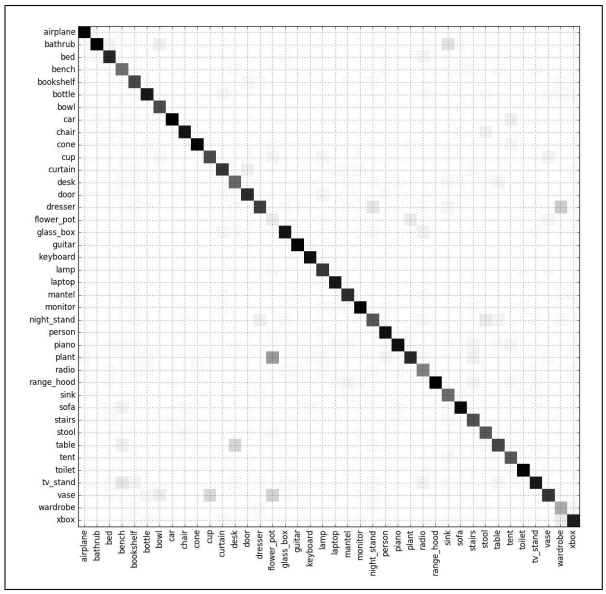

Figure 10. Confusion matrix for a volumetric convolution method (3D convolutions)

The 3D CNNs are stronger (86.7%) than the previously published 3D CNN results on this dataset [13, 22] at 83.1%. However, they don't perform as well as the published work on multiview 2D CNNs. The confusion matrix over the test set is shown in figure 10. Most of the errors are sensible, such as confusing wardrobes for dressers, or flowers pots for plants.

The 2D CNNs, attempting to create a learned embedding layer to meet and surpass the heuristic techniques presented in previous work, perform very well. Their generated images, shown in figures 6 and 7, are seemingly natural. Additionally, their performance (85.5%) is only a hair behind the trained 3D CNN on the same data, and is better than all previously published results with just a single view. That is, DeepPano's 77.6% or even MVCNN's 85.1% on a single image. This suggests that a machine-learned transformation may be superior to an old standby of using a rendering technique, even though the input resolution to the transformation (30x30x30) is very low, compared to a render of the original mesh, as seen in figure 1. This is very impressive considering the ambiguity in the classes of the dataset (wardrobe/dresser, table/tv-stand/xbox) and the low resolution images generated by the model (see figure 6 7. There are many possible ways to mitigate this, including the use of higher resolution voxel grids, or possibly learning an embedding from another, more compact data representation. There is a confusion matrix for a 2D embedding model shown in figure 9. It's errors are very similar to those in the 3D CNN, confusing flower pots for plants and so forth. The biggest difference is that the 2D CNN struggles separating tables from desks, a suggestion that their learned embedding may be very similar (a flat surface). However, even this is an understandably difficult case of semantic labels.

There are many other interesting lessons learned. First, certain models are difficult to learn an embedding for, such as ResNets and GoogleNet. Initializing the upconvolution layers in the embedding with anything other than bilinear interpolation is disastrous. Fine-tuning from an ImageNet model is significantly superior to training from scratch. Another interesting results is these embeddings generate natural enough images that even models without fully-connected classifier layers, such as the network-in-network or GoogleNet models, are able to train very high performing classifiers on the dataset.

All of these results are clearly demonstrated in the table of results on page 7.

5. Conclusions and Future Work

The results in this project show that CNNs have the ability to learn an embedding from 3D to 2D which is superior to the single view render and classification results published in the current state-of-the-art papers [17] [18]. This shows that gradient-trainable transformations across various data formats are very powerful if the goal is only classification loss.

There are many interesting questions left. Such as, do signed distance fields perform better than occupancy voxels? What is the nature of 2D convolutions being more powerful than 3D convolutions? Is it in the model ensemble or is there something inherent in the data? We would like to also explore improved typologies for building an embedding network.

The last line of further work would be to improve the applicability of these models. For example, if we had additional time, we'd like to demonstrate that they can be used on live depth data (from a single viewpoint), using a data source such as a Microsoft Kinect. Additionally, we'd like to perhaps learning a 3D embedding from 2D data. This is the direct inverse of the data transformations explored here and would allow us to perform shape classification from images, using only annotated sets of CAD models.

Architecture	Conv	Embedding	Pretrain	Regularization	Top 1 Accuracy	Iteration
MVCNN 80 Images [18]	2D		ImageNet		90.1%	
MVCNN 1 Image [18]	2D		ImageNet		**85.1%**	
VoxNet [13]	3D		None		83.0%	
DeepPano [17]	2D				77.6%	
3DShapeNets [22]	3D		None		77.0%	
3D CNN	3D		None	None	83.8%	10000
3D CNN	3D		None	Dropout	84.6%	10000
3D CNN + Pooling	3D		None	Dropout	**86.7%**	10000
Embedding + CaffeNet [7]	2D	FC	ImageNet	Dropout	66.5%	3000
Embedding + NetworkInNetwork [11]	2D	FC	ImageNet	Dropout	63.0%	5000
Embedding + GoogleNet [19]	2D	FC	ImageNet		57.0%	5000
Embedding + 1x1 Conv + CaffeNet	2D	FC	ImageNet	Dropout	67.3%	3750
Embedding + 1x1 Conv + CaffeNet (Locked)	2D	FC	ImageNet	Dropout	**75.6%**	4750
Embedding + 1x1 Conv + CaffeNet (from scratch)	2D	FC	ImageNet	Dropout	63.1%	10000
Embedding + ResNet-50 [5]	2D	Upconv1	ImageNet	Dropout	50.3%	10000
Embedding + GoogleNet [19]	2D	Upconv1	ImageNet	Dropout	31.0%	10000
Embedding + VGG[16] + Xavier[4] for Upconv	2D	Upconv1	ImageNet	Dropout	0.03%	10000
Embedding + VGG-16 [16]	2D	Upconv1	ImageNet	Dropout	78.0%	10000
Embedding + VGG-16 [16] + ft	2D	Upconv1	ImageNet	Dropout	**83.5%**	10000
Embedding + VGG-16 [16]	2D	Upconv2	ImageNet	Dropout	**85.5%**	10000

Table 1. Table of results. The results in bold are the best performers when given a single input model. The presented methods in this project are the best in both 2D and 3D CNNs when evaluating just a single image.

References

[1] A. X. Chang, T. A. Funkhouser, L. J. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu. Shapenet: An information-rich 3d model repository. *CoRR*, abs/1512.03012, 2015.

[2] D.-Y. Chen, X.-P. Tian, Y.-T. Shen, and M. Ouhyoung. On visual similarity based 3d model retrieval. In *Computer graphics forum*, volume 22, pages 223–232. Wiley Online Library, 2003.

[3] R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, number EPFL-CONF-192376, 2011.

[4] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International conference on artificial intelligence and statistics*, pages 249–256, 2010.

[5] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[6] G. E. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[7] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the ACM International Conference on Multimedia*, pages 675–678. ACM, 2014.

[8] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[10] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[11] M. Lin, Q. Chen, and S. Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[12] B. C. Lucas, M. Kazhdan, and R. H. Taylor. Springls: a deformable model representation to provide interoperability between meshes and level sets. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2011*, pages 442–450. Springer, 2011.

[13] D. Maturana and S. Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, September 2015.

[14] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.

[15] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and F. Li. Imagenet large scale visual recognition challenge. *CoRR*, abs/1409.0575, 2014.

[16] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[17] S. Song and J. Xiao. Deep sliding shapes for amodal 3d object detection in rgb-d images. *arXiv preprint arXiv:1511.02300*, 2015.

[18] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 945–953, 2015.

[19] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[20] T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

[21] D. J. Watts and S. H. Strogatz. Collective dynamics of small-worldnetworks. *nature*, 393(6684):440–442, 1998.

[22] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1912–1920, 2015.

[23] L. Xu, J. S. Ren, C. Liu, and J. Jia. Deep convolutional neural network for image deconvolution. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 1790–1798. Curran Associates, Inc., 2014.