Investigation: Capsule Nets for Content-Based 3D Model Retrieval

Ryan Lambert
ryan.g.lambert@gmail.com

Abstract

Capsule Networks are a new kind of neural network architecture that are supposed to solve some of the shortcomings of Convolutional Neural Networks (CNN). One of these shortcomings is rotation invariance. CNN architectures generally do not perform well with different orientations unless the network was trained on data that included such rotations. For 3D models, rotation invariance is arguably the most common kind of transformation. Capsule Nets, if able to achieve this, could be a great tool for content based 3D model retrieval. This investigation establishes the current performance of Capsule Nets optimized for 3D models by comparison to the ModelNet benchmark hosted by Princeton.

1 Background

Information retrieval, in practice, centers around the idea of creating an index or semantic space through which things can be compared. This can be sentences, images, 3D models, and DNA sequences. Through the use of recommendation engines and search engines we take hold of this value almost every day.

The value information retrieval adds is in recommending relevant documents of which there are too many for the average human to parse through.

As access to CAD tools and things like 3D printers increases, content based 3D model retrieval will have great value. Less experienced designers can expose themselves to relevant design patterns of people many years their senior much like how a programmer will trall Stack Overflow for better design patterns or solutions to a problem.

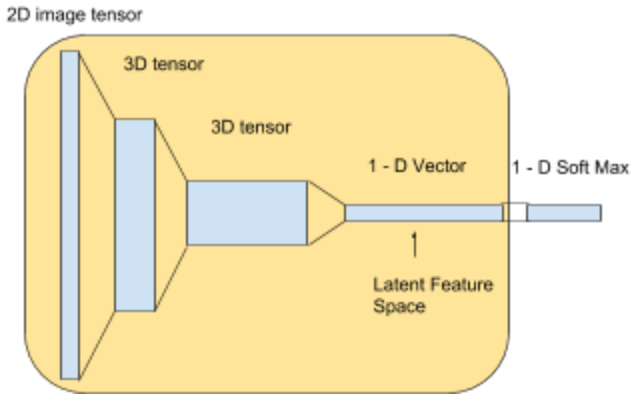
Research in Content based 3D model retrieval is a few decades old. Old methods of content-based 3D model retrieval tend to use more manually engineered features: transforms that focus on frequency of surface change[3], steps required to assemble [2] (from the world of machining). Most newer methods are using some combination of CNNs. [4, 1]

Recently, (~2015) a benchmark was established and hosted by Princeton called ModelNet. [1] It consists of two datasets of labeled 3D models. ModelNet10 is a 10 class labeled dataset. ModelNet40 is a 40 class labeled dataset.

Neural Networks for Information Retrieval

Most of us are familiar with neural networks telling us if something is a kitten or a hotdog, but both supervised and unsupervised neural networks have also been used for information retrieval. The use differs slightly. The problem is no longer “is this a kitten.” Instead, you have a query, and the query is “what is most similar to this kitten?”

Doing this with a neural network requires a slight modification.



For the supervised model, you can keep all of the layers of the network except the very last layer where you output probabilities for classification. It's like slicing the part of the network that contains the most distilled representation for that image (a 1D vector). Basically you can think of this network as a transformer that moves an image (or model, or text) into a 'latent feature space'. You can cut from the input layer to any layer in the network and achieve different kinds of feature spaces. I'll use every layer except the last (the softmax probabilities vector) for this paper.

2 Problem

CNNs aren't completely translation invariant, and aren't rotation invariant.[5] The current best performer on ModelNet[1] called PANORAMA-ENN uses an ensemble of CNNs that have been trained on aligned models and data augmented via labeled rotations.

In practical terms, if you want a CNN to be able to understand that there's an upside down helicopter in a picture then that network would need to be trained on data containing upside down helicopters. That's what PANORAMA-ENN did for many angles of models on ModelNet10 and ModelNet40.

Augmenting data is common in deep learning. However, for information retrieval, you're already trying to index a lot of data. So there's good reason to avoid the use of augmented data for information retrieval.

Capsule Nets allegedly solve the rotation / translation issue via 'dynamic routing by agreement'. [8]

This paper will investigate performance of 3D convolutional Capsule Nets. I will be using some code implemented by Xifeng Guo [7] which was implemented for MNIST. The biggest difference is in applying this to 3D voxel data instead of 2D images.

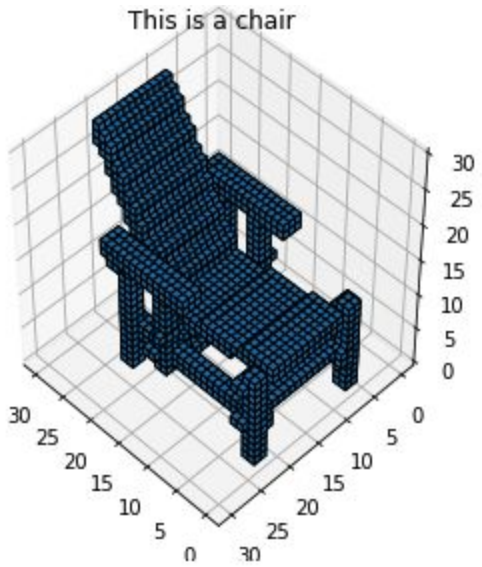
The ModelNet benchmark contains a 10 class and 40 class dataset. Accuracy is used to measure classification performance. Mean Average Precision is used to measure information retrieval performance.

3 Analysis

3.1 Data

Samples	ModelNet10	ModelNet40
Train	3991	9832
Test	908	2468
Validation	10% of Train	10% of Train
Classes	10	40
Dimension	30x30x30	30x30x30
Aligned	Yes	No

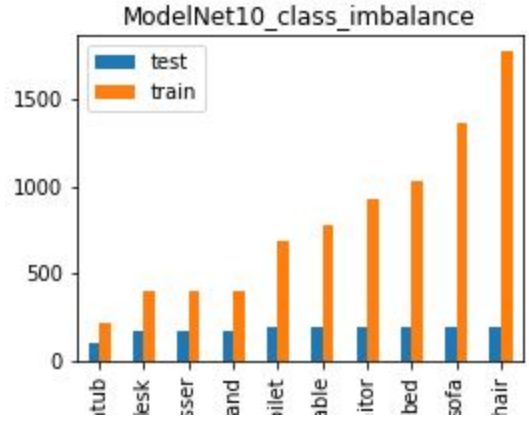
Each sample is a file in *.off format that represents coordinates of a polygonal object. I've transformed these files from *.off format to *.binvox format (a 3D tensor).

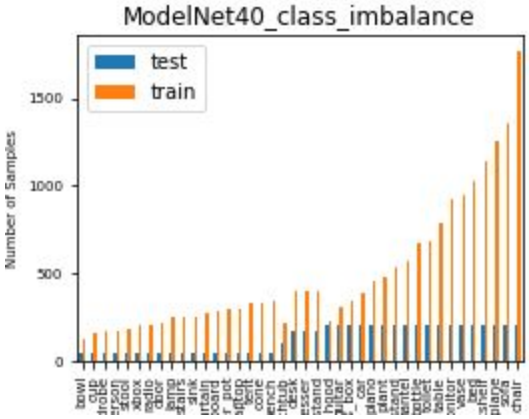


(see appendix: Data Preprocessing)

3.2 Class Imbalance

There is class imbalance to account for.





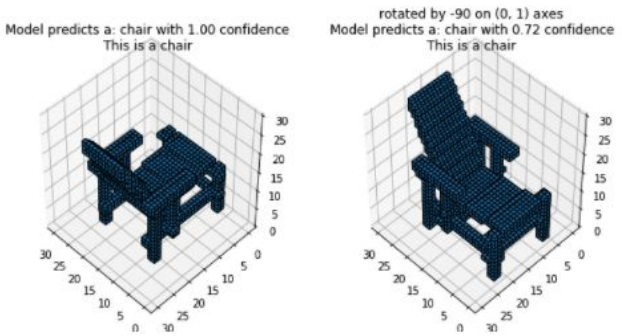
To handle the class imbalance the validation set is created via stratified shuffle sample from the training data. Shuffled is at random while stratified creates the same representation of classes found in training set in the validation set.

Each class in the training set is upsampled to a number of samples equal to whatever the highest sample count class is. To see verification plots please see the appendix.
(see appendix: 'Class Balance')

3.3 CNNs Rotational Variance

To illustrate the problem that CNNs have with rotational variance I trained a CNN similar to what was created in "3D ShapeNets: A Deep Representation for Volumetric Shapes." [10]

In the picture on the left is a chair how it is normally aligned in the ModelNet10 test set (which is aligned). The confidence is ~100% that this is a chair while the confidence for the chair on the right is 72%



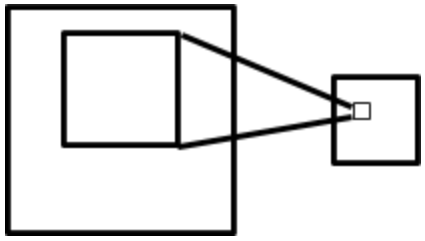
(see appendix: CNN Rotation)

If rotational invariance exists, then the confidences should be essentially the same. They are not.

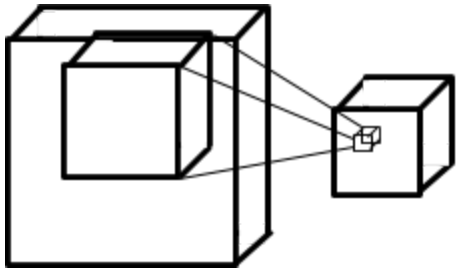
3.4 Capsule Nets

The implementation of Capsule Nets for 3D voxel data was based on CapsNet-Keras [7], a Capsule Net implementation designed for the MNIST digits classification benchmark and others. *(see 3d_model_retriever/modelnet40_arch.py and 3d_model_retriever/modelnet10_arch.py)*

The first Conv2D layer and the Conv2D layer inside of only convolutional capsule layer are changed to Conv3D layers.

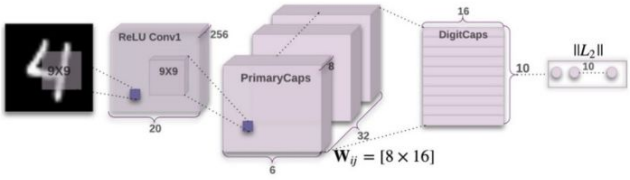


(2D Convolutional Layer)



(3D Convolutional Layer)

The main difference between Capsule Nets and CNNs is in the way outputs are passed from one layer to the next. In a convolutional layer all information is passed forward. If you use max pooling in a CNN layer then only those neurons which are maximally activated pass their outputs forward. For Capsule Nets, each capsule's output is conditioned on the idea of 'routing by agreement'. In practice this means that the capsule is only passing outputs forward for entities in the image for which it's certain actually exist regardless of pose/orientation.



Due to time constraints, evaluating other 'multiple convolutional capsule layers' architectures will require future work to implement. *(see appendix: Complications)*

Algorithm	ModelNet40 Classification (Accuracy)	ModelNet40 Retrieval (MAP)	ModelNet10 Classification (Accuracy)	ModelNet10 Retrieval (MAP)
Minto et al.[33]	89.3%		93.6%	
RotationNet[32]	97.37%		98.46%	
LonchaNet[31]			94.37	
Achlioptas et al. [30]	84.5%		95.4%	
PANORAMA-ENN [29]	95.56%	86.34%	96.85%	93.28%

(As a frame of reference see appendix: Benchmark)

3.5 Benchmark

"Out of the box" Capsule Networks:

	ModelNet10	ModelNet40
Accuracy	**9%**	****Similar or worse, saving model training time.*
Rotated 90	*9%*	
MAP	**19%**	
Rotated 90	*19%*	

Accuracy

Classification accuracy, a straight-forward metric, shouldn't require any further explanation.

Mean Average Precision (MAP)

The information retrieval portion of the benchmark is assessed with MAP - Mean Average Precision. The motivation for this metric is based on the difference in the objective for information retrieval. Information retrieval having to do with accurately ranking results for a query.

4 Methodology

4.1 Data Preprocessing

Each sample is its own file in *.off format. These files consist of a set of coordinates and indices for those coordinates which describe a polygon. In order to be useful for performing 3D convolutions upon, the polygons must be converted to voxels.

(see appendix: Data Preprocessing)

4.2 Implementation

The Capsule Net architecture in Capsnet-Keras is tuned for digits. Our challenge is to find the optimal parameters for 3D data in ModelNet instead of that found in MNIST digits.

Calculating Mean Average Precision

$$\mathrm{MAP}(Q) = \frac{1}{|Q|}\sum_{j=1}^{|Q|}\frac{1}{m_j}\sum_{k=1}^{m_j}\mathrm{Precision}(R_{jk})$$

In classification the goal is to be correct. But, in information retrieval, the goal is to have ranked all of the relevant documents to a given query correctly. This is where the motivation of Mean Average Precision comes from. Results earlier in the rank carry more weight in the MAP score than later ones.

Mean Average Precision is simply the mean of the aggregated average precision calculated for each query. Each sample is used as a query, 100% precision would mean that all samples of the same class were retrieved and ranked at the top of the

result with higher ranks carrying more weight than lower ones.

Example:
If there are 2 chairs (relevant items) to a chair query and you rank all of them in ranks:
($chair_1$, $chair_2$, ..., not_chair),
then your average precision for this chair query and resultant ranked list will be ***100%.***

If instead you ranked the chairs as:
(not_chair, $chair_1$, $chair_2$, ..., not_chair)
All chairs being in rank 2 and 3.
Then your average precision for this chair and resulting ranked list would be:
58.3%
(see code: `3d_model_retriever/utils.py:39`)

From the example, the Mean Average Precision would be the mean of all the results of average precision for each query. (for each sample there is a query).

Implementing Conv3D

There are two layers where I'm changing `Conv2D` to `Conv3D`. The first change is within the regular convolutional layer (located in each 'modelnet arch' file). The second change is the convolutional layers within the capsule layer itself.

- `3d_model_retriever/capsulelayers.py:178`
- `modelnet10_arch.py:72`
  `modelnet40_arch.py:70`

Metrics

One interesting part of the project was the need to process metrics after model creation. In a sort of 'eager' fashion, the metrics and plots are created whether or not I will actually use them. After each model is created, a 'process results' function is called to calculate all the metrics ahead of time leaving less processing come time for analysis.

- Precision recall curves, confusion matrices, and precision recall auc are all used to evaluate the discriminative power of the network
- Latent space matrix for info retrieval metric Mean Average Precision
- Save y_prediction for ad-hoc analysis if necessary

Complications

Exploring different Capsule Net specific architectures will have to be a future endeavor. Capsule Nets are pretty convoluted. No pun intended. Due to time constraints, my understanding is somewhat limited in how to implement more robust architects in Keras. This paper will focus solely on optimizing the existing Capsule Net architecture for 3D models with a grid search.

Issues with checkpointing and saving pickled models wasted a little over 12 hours of development time. Checkpointing in keras is non-trivial when training across multiple GPUs. Saving pickled models with custom layers required some adjustments to Capsnet-Keras code. [7]
See 3d_model_retriever/capsulelayers.py:99

Robustness

For the sake of training time, I used single hold out validation. Finally, upon choosing a model, I perform a 5-fold stratified shuffled cross validation validate our model accuracies and mean average precision scores.

4.3 Refinement

Capsnets out of the box performance

I stopped this model early on since the validation accuracy never went above 10% during training.

Capsnet ModelNet10 Grid Search

After tuning Capsule Nets on ModelNet10 with hunches I used a grid search to get to an optimal set of hyper parameters. With more time (and money) there is undoubtedly a more optimal solution.
(see appendix: Grid Search ModelNet10)

Capsnet ModelNet40

A similar process was applied to ModelNet40. However, given the constraints of having to use the same model on ModelNet40 and ModelNet10, it was faster to perform design iterations on ModelNet10 data and then try to get that model to run on ModelNet40. Minimal grid search was done on ModelNet40. Further search is certainly warranted, especially after accidentally discovering how helpful random restart was for ModelNet40.

After seeing the positive effect of random restarts in avoiding local minima, I started every model training of ModelNet40 with random restarts on a small batch of 500 samples when training accuracy doesn't go above a threshold of .4. Once warmed up the network switches over to a batch size of 128/256 for 50 Epochs or early stop with decreasing learning rate.
(see appendix: Grid Search ModelNet40)

5 Results

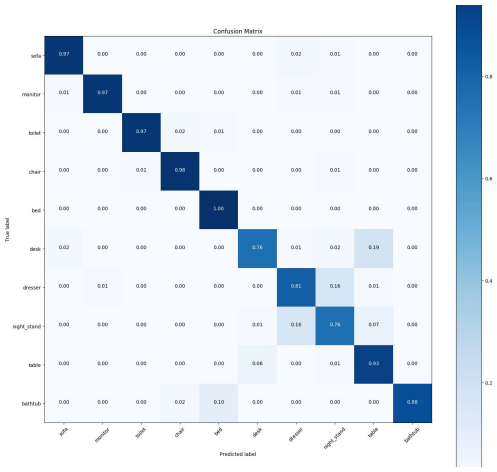
5.1 Evaluation / Validation

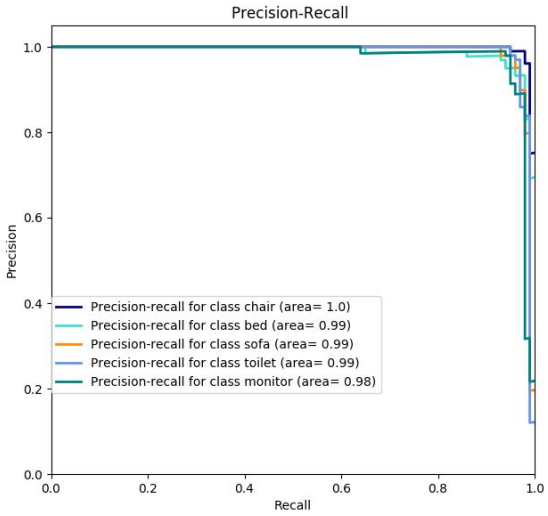
Performance

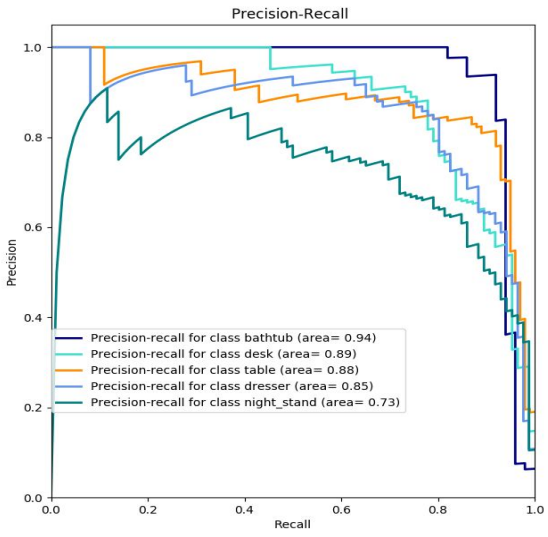
	ModelNet10		ModelNet40	
	score	2*stderr	score	2*stderr
Accuracy	**93.08%**	1.18%	**82.73%**	2.12%
Rotated 90	*31.31%*	*1.68%*	*41.09%*	*4.27%*
MAP	**88.44%**	4.15%	**70.10%**	2.22%
Rotated 90	*35.10%*	*3.75%*	*30.49%*	*2.24%*

Scrutinizing Discriminative Properties

From the precision recall plots and the confusion matrix the least performant class "night stand" gets mixed up with "dresser" pretty frequently (20% of the time). Comparing dimensional perturbations(next page) for a true positive "night stand" with low confidence and a false negative "night stand" the dimensional perturbations should be similar(next page).

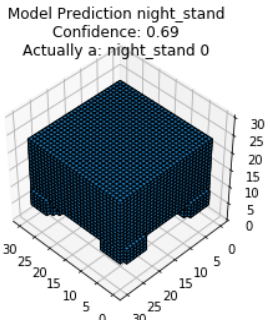


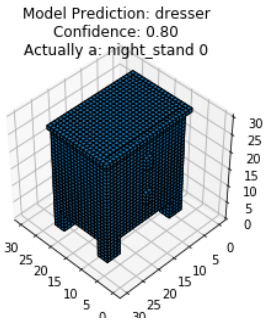




If there is discriminative power then the dimensional perturbations will not all be the same.
If there is low discriminative power, then more of the dimensional perturbations will be the same.

Low Discriminative Power

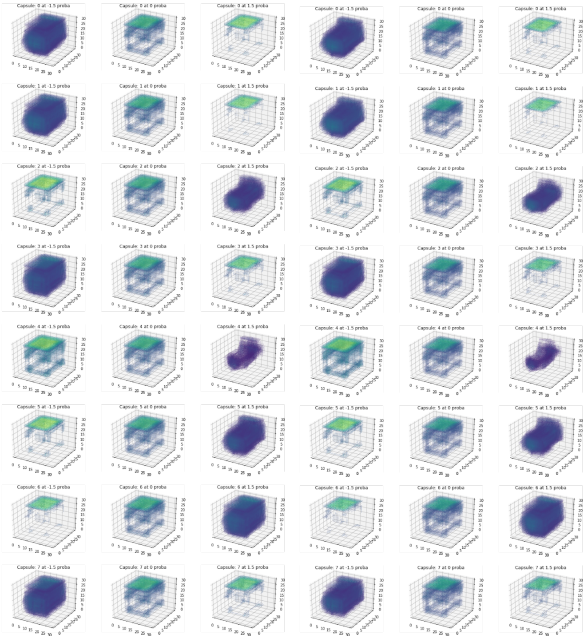




True Pos

False Neg

1 2 3 1 2 3

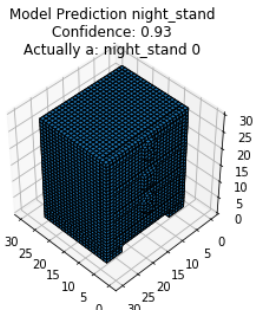


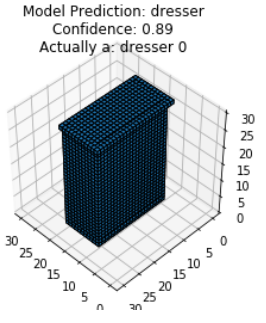
(for full size see appendix: Dresser vs Nightstand)

Conversely, when there is high discriminative power we should see dimensional perturbations that are more obviously different between the two objects. (when the model is right and it was confident)

Column 2 is simply the activation map for that class. You'll notice these are exactly the same for the Low Discriminative Power. Notice for the High Discriminative Power, you'll see that they're different.

High Discriminative Power

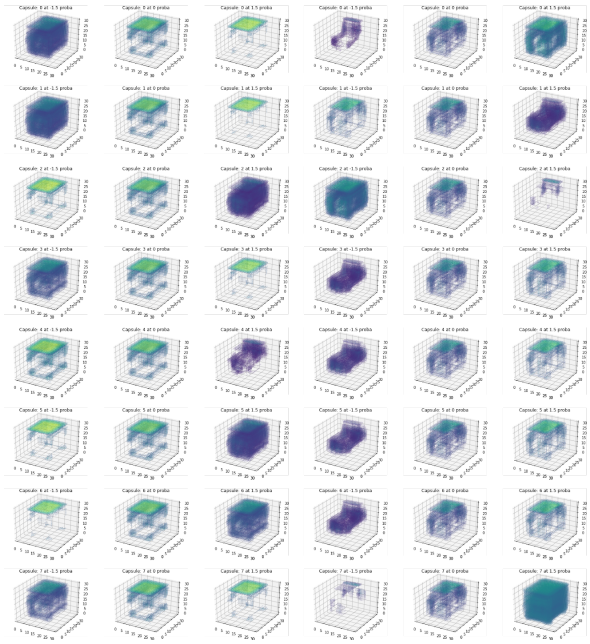




True Positive

True Negative (dresser positive)

1 2 3 1 2 3

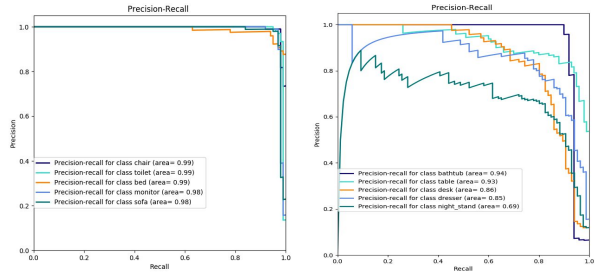


(for full size see appendix: Night Stand True Positive and True Negative…)

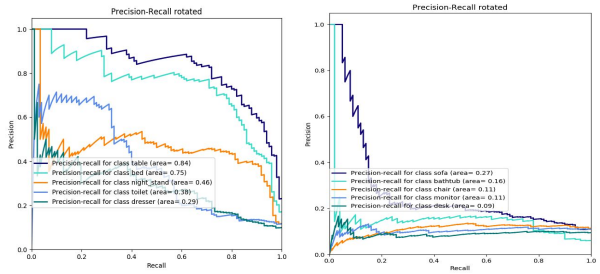
Columns 1 and 3 describe how the activations change along that capsule dimension. Between the classes you'll notice that these are significantly different from one another.

Rotational Invariance Not Achieved

ModelNet10 PR AUC



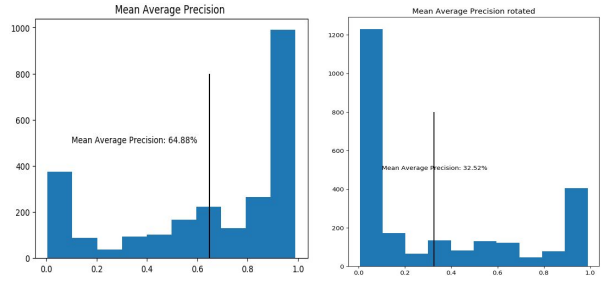
ModelNet10 PR AUC Rotated



These two pairs of charts show that we clearly did not achieve rotation invariance for ModelNet10.

Perhaps more easily seen with a plot of the distributions of average precisions.

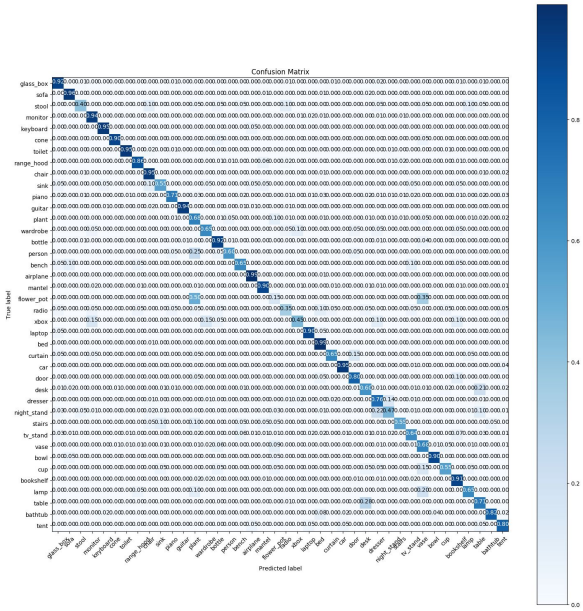
Not Rotated Rotated



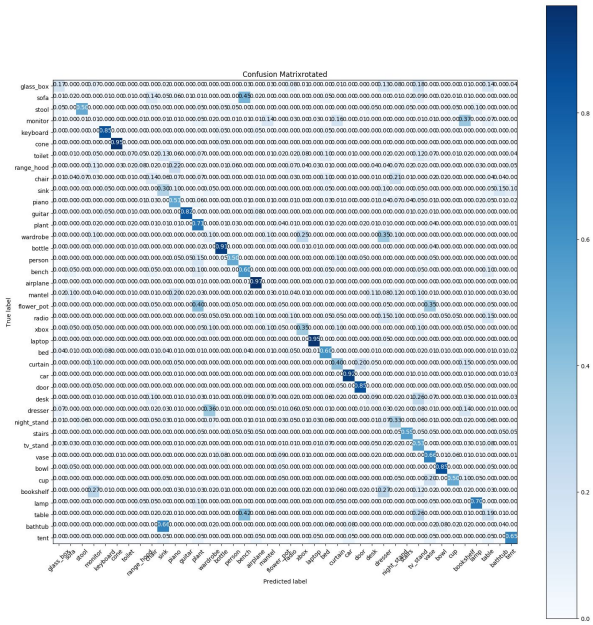
ModelNet40 PR AUC

For ModelNet40 Rotational Invariance will be more easily seen by looking at differences in confusion matrices

Not Rotated

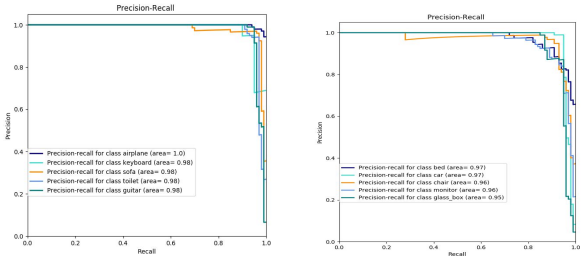


Rotated

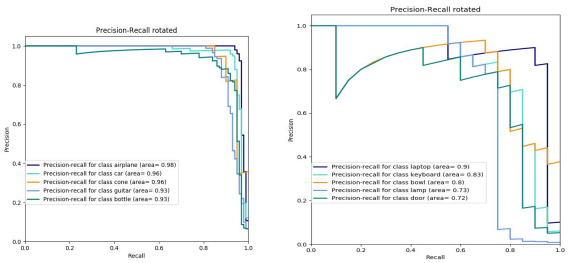


For rotated data, there are quite a few dark colored squares along the diagonal, signifying successful classification in spite of rotation. Taking a closer look at the top ten performing classes PR AUC curves below

Not Rotated

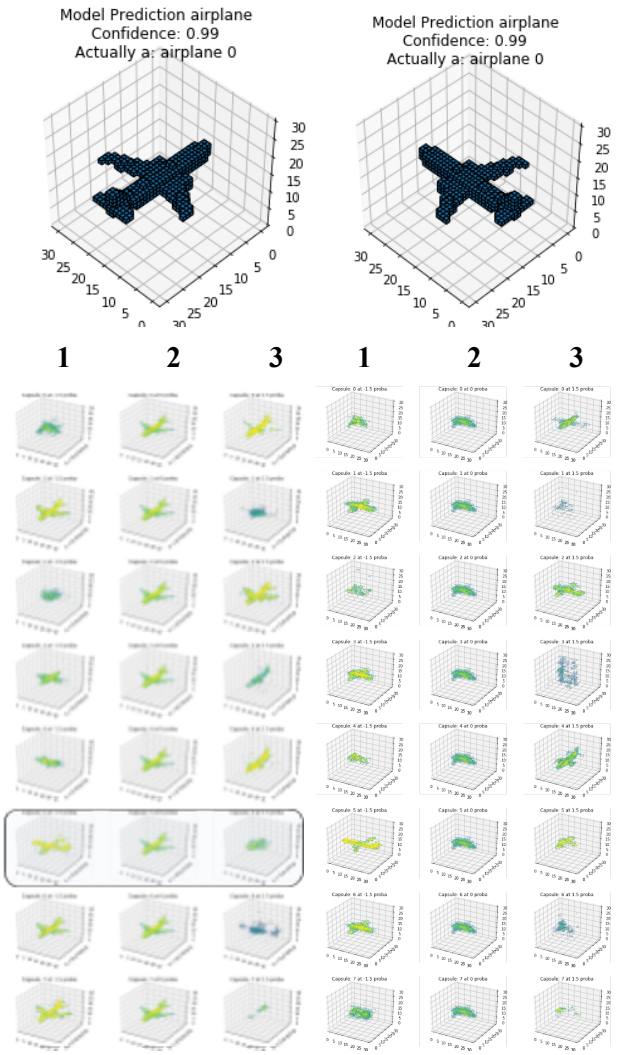


Rotated

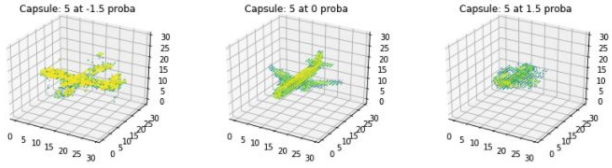


At first glance it appears we may have achieved *some* rotational invariance for ModelNet40? At least the top 5 to 10 AUC classes?

Class	AUC rotated	Make Sense?
Airplane	.97	no
Car	.97	no
Cone	.96	yes
Guitar	.96	Maybe the neck?
Bottle	.95	yes
Laptop	.9	closed
Keyboard	.83	maybe
Bowl	.8	yes
Lamp	.73	yes
Door	.72	With enough training data not in the same plane



Upon deeper investigation it appears that the 3rd capsule from the bottom has learned another orientation for a plane! **See left column 1 row 6**



(For more please see appendix: ModelNet40 Rotation Invariance and ModelNet40 Confusion Matrix and Airplane)

For the ModelNet40 models where there was rotational invariance, it was for models whose discriminative features were inherently rotationally equivariant compared to the other classes. (e.g. cones, bottles, guitars (the leading from the body to the neck).

5.2 Justification

The accuracy and mean average precision are nothing to sneeze at. However, the intention of considering Capsule Networks was for their rotation invariance. That was not achieved. So I can't recommend the use of this Capsule Network architecture for 3D Models. It is possible other architectures might behave better.

6 Conclusion

6.1 Performance Against Benchmark

"Out of the box" Capsule Networks:

	ModelNet10	ModelNet40
Accuracy	**9%**	
Rotated 90	*9%*	
MAP	**19%**	
Rotated 90	*19%*	

Capsule Nets with optimized hyper parameters

	ModelNet10		ModelNet40	
	score	2*stderr	score	2*stderr
Accuracy	**93.08%**	1.18%	**82.73%**	2.12%
Rotated 90	*31.31%*	*1.68%*	*41.09%*	*4.27%*
MAP	**88.44%**	4.15%	**70.10%**	2.22%
Rotated 90	*35.10%*	*3.75%*	*30.49%*	*2.24%*

There is clearly a vast improvement in the results from the initial model. In spite of not building a rotationally invariant Capsule Network, I did achieve somewhat successful model optimization.

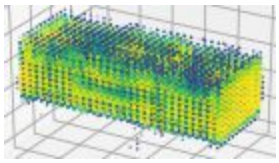
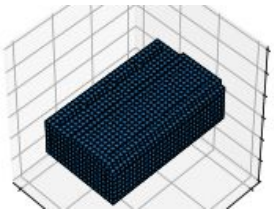
7.2 Reflection

Plotting and Visualizations Tooling

Right after the acceptance of the proposal my goal was to simply get a plot of a voxel array of one of the *.off files.

The source of the ModelNet data had tools that were written in Matlab to handle these *.off files but not in Python. After some searching I found 'binvox' which is written in C and run from the command line. The author of binvox made reference to some libraries written in various languages, one of them in python call 'binvox_rw_py'[13], used to convert *.binvox files into numpy arrays.

Plotting the array in voxel format using Matplotlib is incredibly slow. As a consequence, some of the major plots in this report aren't actually using voxels, but scatter plots where the size and intensity of the dot represent how activated that area is.

 vs. 

Exploratory visualizations should be quick and simple. The dots do the trick.

The initial plots looked a lot cloudier. It wasn't as useful to see one big solid cloud so I added two arguments that amplify the size change of the dots and another that ignores dots under a threshold. The threshold has an effect of "carving away" low activated points.

This moved my visualizations from cloudy and non-interpretable to something that looks like it was 'carved' out of that cloud.

Organization

Later into the project it became clear I needed to separate the plotting of PR curves, confusion matrices, and average precision histograms, to save time and money training these models.

The basic idea was to save all of the results for each model training to a unique directory like so:

/results
  /model_<name>_acc_<accuracy>_map_<map>
    /precision_recall_plots/

Precision_recall.png
Precision_recallrotated.png
/confusion_matrix.png
/models
<each_model>.hdf5
/latent_space.npy
/details.csv (<< more details about model)

Most of the time I won't even touch the models I end up saving. But they're there just in case I need further exploration.

Random Restarts / Warm up sets

I did this manually on ModelNet40. However, when doing cross validation this isn't really possible to do manually.
I stumbled upon this github issue discussing the matter:
(https://github.com/keras-team/keras/issues/609) and I found a way to do it here:
(https://www.codementor.io/nitinsurya/how-to-re-initialize-keras-model-weights-et41zre2g).

Rotational Invariance

I had initially shown anecdotal examples where CNNs don't do well with rotational variance. *(see appendix: CNN Rotation Issue)*

After seeing that Capsule Nets anecdotally were also not performant on rotated objects, it seemed prudent to create a simple rotated test set to display this fact to the reader. Not all of the results in the 'results' folder will contain scores for rotated models. However, enough of them do show these scores for you to see that rotational invariance wasn't achieved.

Interpretability

Dimensional perturbations, as used by Hinton et al. [12], are implemented already in CapsNets-Keras[7]. This attachment point to the Capsule Network has the variable name 'manipulate_model'.

To investigate low performing classifications and suspicious results, as you saw in section 5, I used dimensional perturbations. Hinton et al did this on MNIST digits with CapsNets. For MNIST they extracted things like "italic-ness" or "stroke thickness" [12]

This is a nice feature specific to capsule nets that makes them easier to interpret.

Time and Cost

Start: 3/5/2018
Finish: 4/2/2018
Devotion: full time+
Cost: ~$380
EC2:
p2.8xlarge: 38 hrs
p2.xlarge: 93 hrs
p3.2xlarge 8 hrs
g3.8xlarge 6 hrs

7.3 Improvement

Hyper Parameter Optimization with Randomized Search

Evaluating the best parameters to use is time consuming. While working on this paper I discovered Random Search for Hyper-parameter Optimization [11]. This would speed up the process of testing out an architecture while not having to worry whether you've truly searched the hyper-parameter space comprehensively enough.

Evaluate Different Architectures

We didn't do what is normally done with neural networks. Namely, we didn't "Go Deeper".
I'm deeply suspicious of possible success with either or both of deep architectures and including the use of EM Capsules. [13]

References

[1] http://modelnet.cs.princeton.edu/

[2] Antonio Cardone, Satyandra K. Gupta, Abhijit Deshmukh, Mukul Karnik
Machining Feature–based Similarity Assessment Algorithms For Prismatic Machined Parts
http://terpconnect.umd.edu/~skgupta/Publication/CAD06_Cardone_draft.pdf

[3] Antonio Cardone, Satyandra K. Gupta1 , and Mukul Karnik
A Survey of Shape Similarity Assessment Algorithms for Product Design and Manufacturing Applications
http://terpconnect.umd.edu/~skgupta/Publication/JCISE03_Cardone_draft.pdf

[4] K Sfikas, I Pratikakis and T Theoharis, Ensemble of PANORAMA-based Convolutional Neural Networks for 3D Model Classification and RetrievalComputers and Graphics

[5] https://youtu.be/rTawFwUvnLE?t=31m18s

[6] Multi Armed Bandit Sampling, Benjam´ın Guti´errez, Lo¨ıc Peter, Tassilo Klein, Christian Wachinger
https://arxiv.org/pdf/1705.08111.pdf

[7] Xifeng Guo, CapsNet-Keras
https://github.com/XifengGuo/CapsNet-Keras

[8] Sabour, Frosst, Hinton, Dynamic Routing Between Capsules, https://arxiv.org/pdf/1710.09829.pdf, page 2

[9] Patrick Min, Binvox, http://www.patrickmin.com/binvox/

[10] 3D ShapeNets: A Deep Representation for Volumetric Shapes,
http://vision.princeton.edu/projects/2014/3DShapeNets/paper.pdf

[11] Random Search for Hyper-Parameter Optimization, James Bergstra JAMES.BERGSTRA@UMONTREAL.CA Yoshua Bengio, http://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf

[12]Geoffrey E. Hinton, Sara Sabour, Nicholas Frosst,
Dynamic Routing Between Capsules, page 4, 6, https://arxiv.org/pdf/1710.09829.pdf

[13]Geoffrey E. Hinton, Sara Sabour, Nicholas Frosst,
Matrix Capsules with EM Routing, https://openreview.net/pdf?id=HJWLfGWRb

[14] binvox_rw_py, https://github.com/dimatura/binvox-rw-py

Appendix

Data Preprocessing

Each sample in the data is represented by an *.off file. These coordinates and indices describe the vertices of a polygon. Since we're interested in working with voxels (a 3D tensor), I'm using a tool called binvox to convert these polygons into voxels which are stored as *.binvox files.

*.off File Example of a cube	Explanation
OFF 8 6 0 -0.500000 -0.500000 0.500000 0.500000 -0.500000 0.500000 -0.500000 0.500000 0.500000 0.500000 0.500000 0.500000 -0.500000 0.500000 -0.500000 0.500000 0.500000 -0.500000 -0.500000 -0.500000 -0.500000 0.500000 -0.500000 -0.500000 4 0 1 3 2 4 2 3 5 4 4 4 5 7 6 4 6 7 1 0 4 1 7 5 3 4 6 0 2 4	<header> <num vertices>, <num faces>, <num edges> <coordinates> .. .. Face index <num sides>, <index1>, …, <index_n> .. The indices correspond to which of the coordinates are used for this face

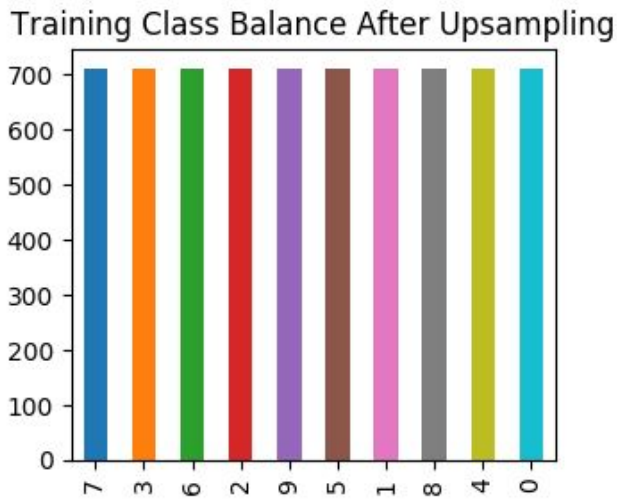
Each sample is transformed from *.off file to a voxel representation as a *.binvox file. This is accomplished with a tool called Binvox[9] which is wrapped by my `3d_model_retriever/binvox_converter.py`.
Reading a binvox file is performed using https://github.com/dimatura/binvox-rw-py.

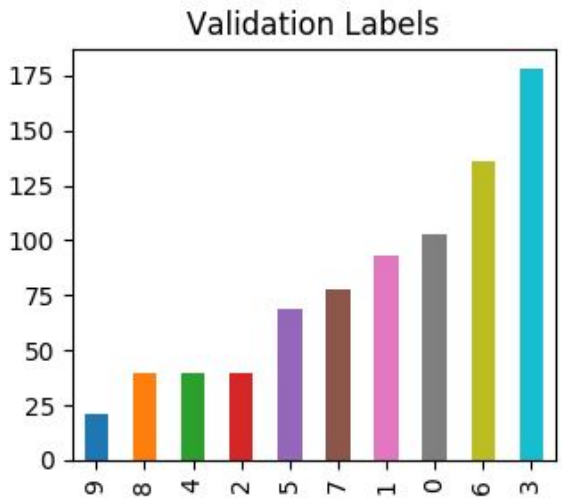
Complications

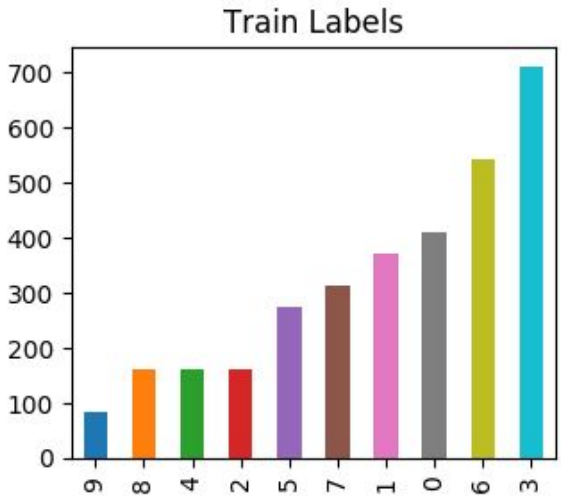
- Saving Keras models with capsnet custom layers was non-trivial to troubleshoot (full day). It was a surprise. But, then I hadn't really used Keras much before now. I ended up overloading the `get_config` method for CapsuleLayer so that keras would save it's configuration properly. (https://github.com/keras-team/keras/issues/4871)
- Checkpointing appears to be somewhat broken for Keras when in multi-GPU mode (https://github.com/keras-team/keras/issues/8123). Because of this I ended up using the two callbacks that do 'early stopping' and 'learning rate change' to find an optimal model.
- Since my intent is to achieve a tangible result in 3 to 4 weeks, doing a respectable analysis on different Capsule Net architectures will be a future endeavor. Capsule Nets are quite different from CNNs making it a bit hard for me to efficiently try new architectures within Keras. For now I'll be using similar architecture to what Hinton used against MNIST, but with different hyperparameters.
- Random Restarts and Warmup:
  I stumbled upon this github issue discussing the matter: (https://github.com/keras-team/keras/issues/609) and I found a way to do it here:
  (https://www.codementor.io/nitinsurya/how-to-re-initialize-keras-model-weights-et41zre2g).

Class Balance

Top Right vs Bottom left is a visual test that I've sampled the validation data in a way that is representative of the training data.
Top Left is a visual 'test' that I've successfully upsampled the training data.







CNN Rotation Issue

This is the architecture of the CNN following the example given by:
http://vision.princeton.edu/projects/2014/3DShapeNets/paper.pdf

There are some minor differences as the goal was to illustrate the **effect of rotational variance** and not necessarily replicate the entire paper! :-D

Layer (type)	*Output Shape*	*Param #*
conv3d_1 (Conv3D)	*(None, 15, 15, 15, 48)*	*10416*
conv3d_2 (Conv3D)	*(None, 8, 8, 8, 160)*	*960160*
conv3d_3 (Conv3D)	*(None, 8, 8, 8, 512)*	*5243392*
flatten_1 (Flatten)	*(None, 262144)*	*0*
dense_4 (Dense)	*(None, 1200)*	*314574000*
dropout_1 (Dropout)	*(None, 1200)*	*0*
dense_5 (Dense)	*(None, 4000)*	*4804000*
dropout_2 (Dropout)	*(None, 4000)*	*0*
dense_6 (Dense)	*(None, 10)*	*40010*

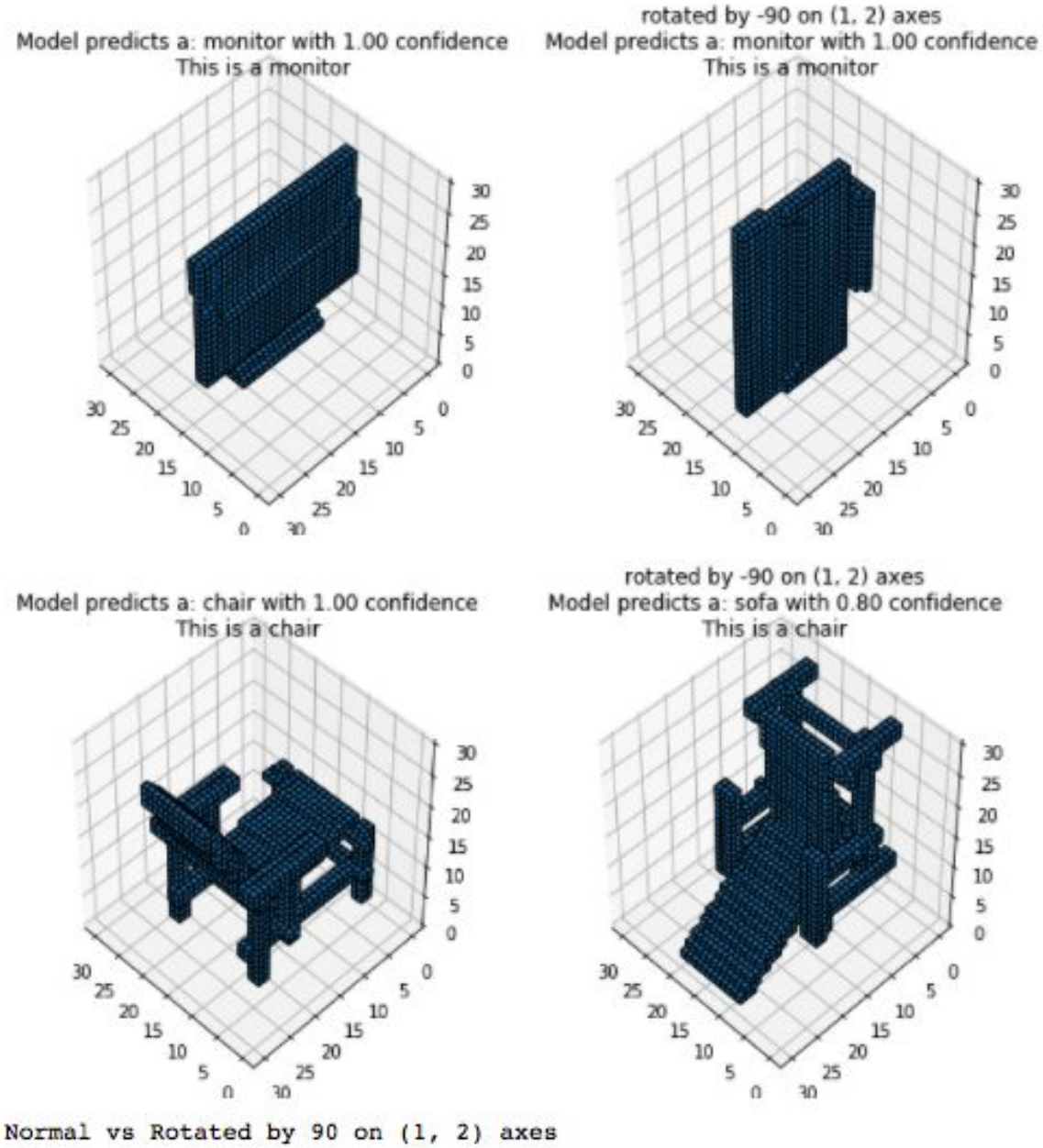
Total params: 325,631,978
Trainable params: 325,631,978
Non-trainable params: 0

Note: Max Pooling was not used by the authors of the 3D Shape Nets paper [10]. CAD models are almost always centered in the frame of reference. ModelNet10 and 40 were no exception. I believe this was also their reasoning.

CNN Rotation Issue

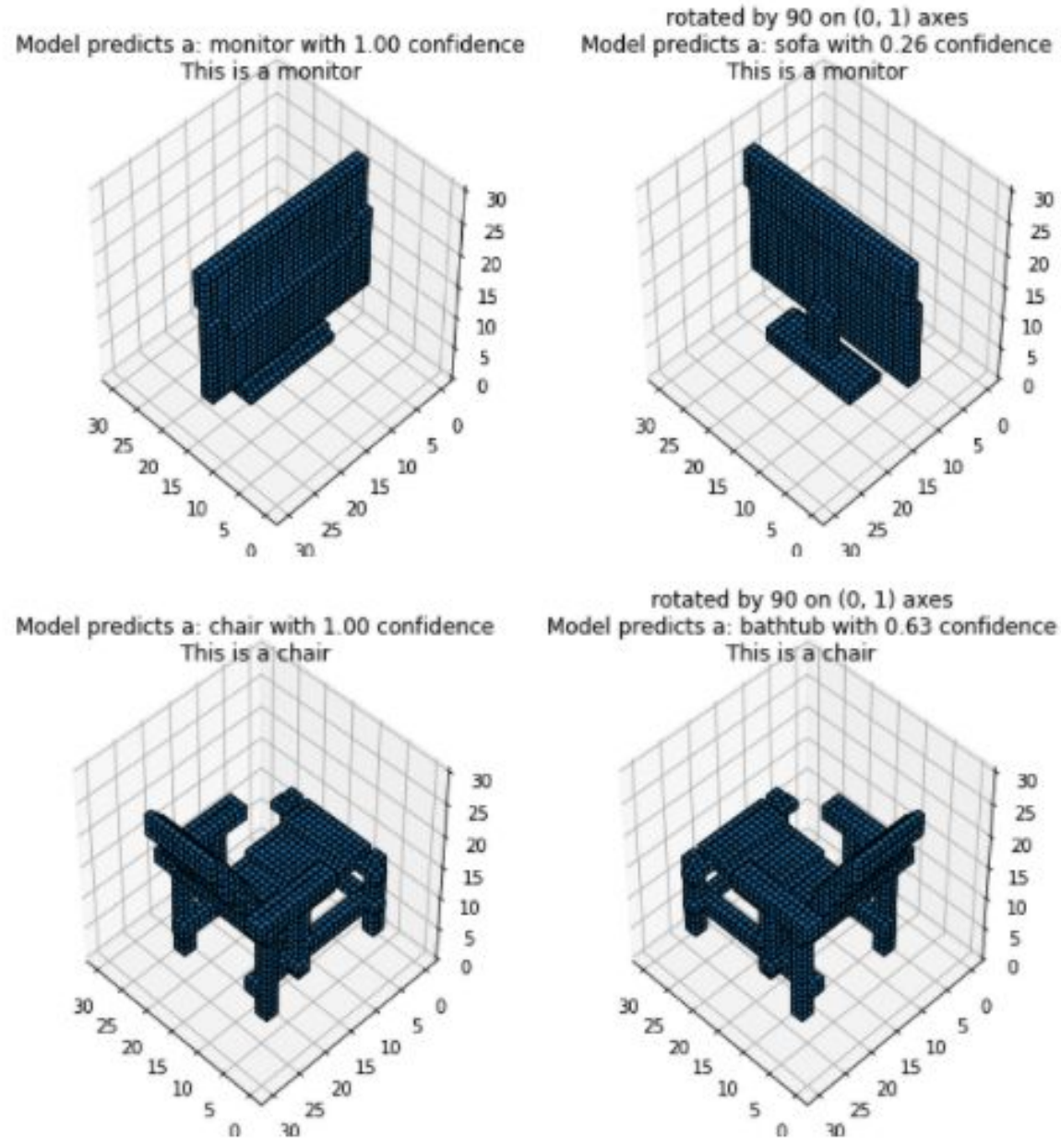
The important thing to notice is the difference in prediction confidence (especially when it's wrong).

The monitor score is right on even when rotated, however…

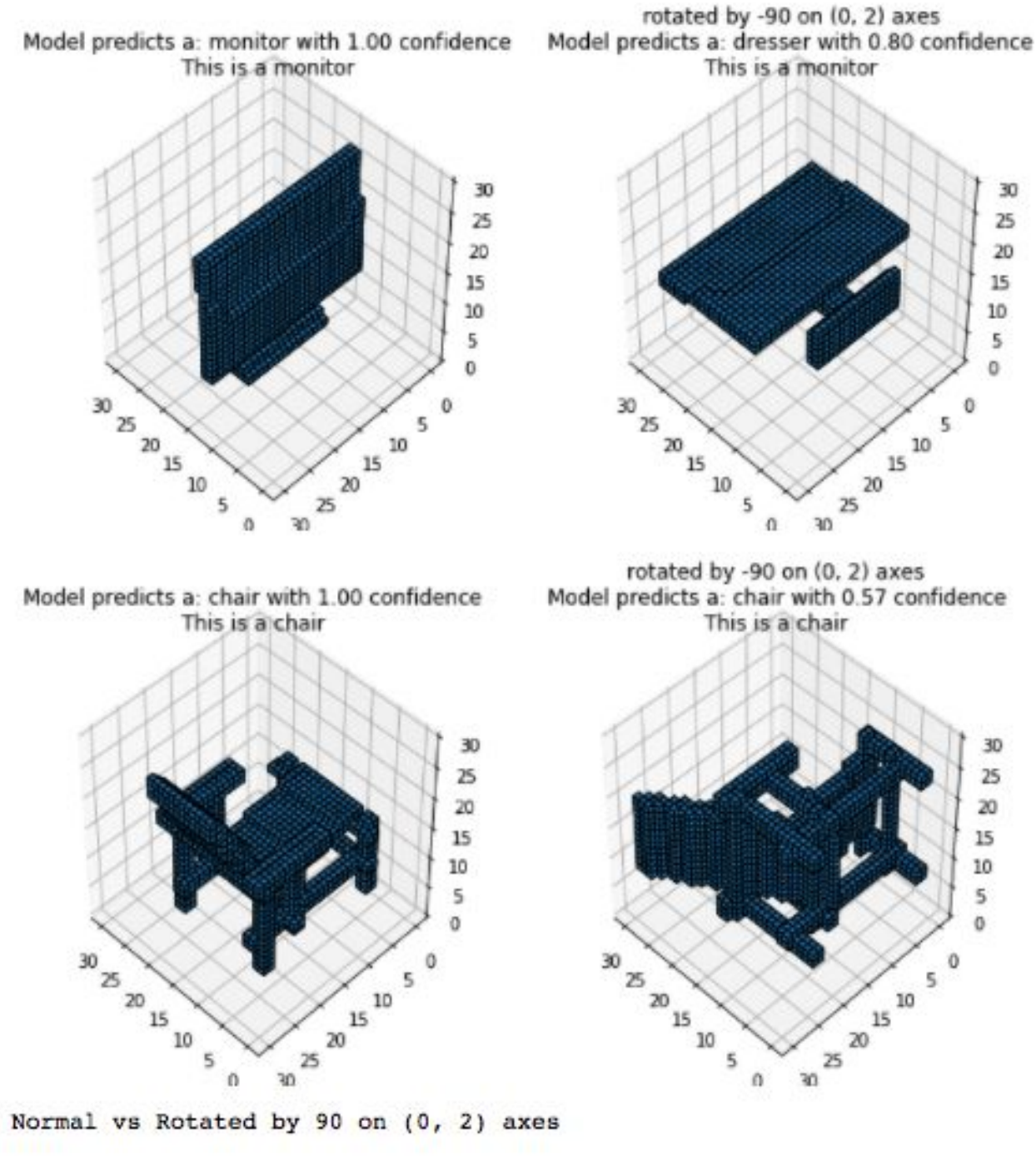


CNN Rotation Issue

This monitor is rotated and clearly outside of the body positioning that the model expects. It guessed that this monitor is a sofa with .26 confidence. It's nice that the confidence is lower though.



CNN Rotation Issue

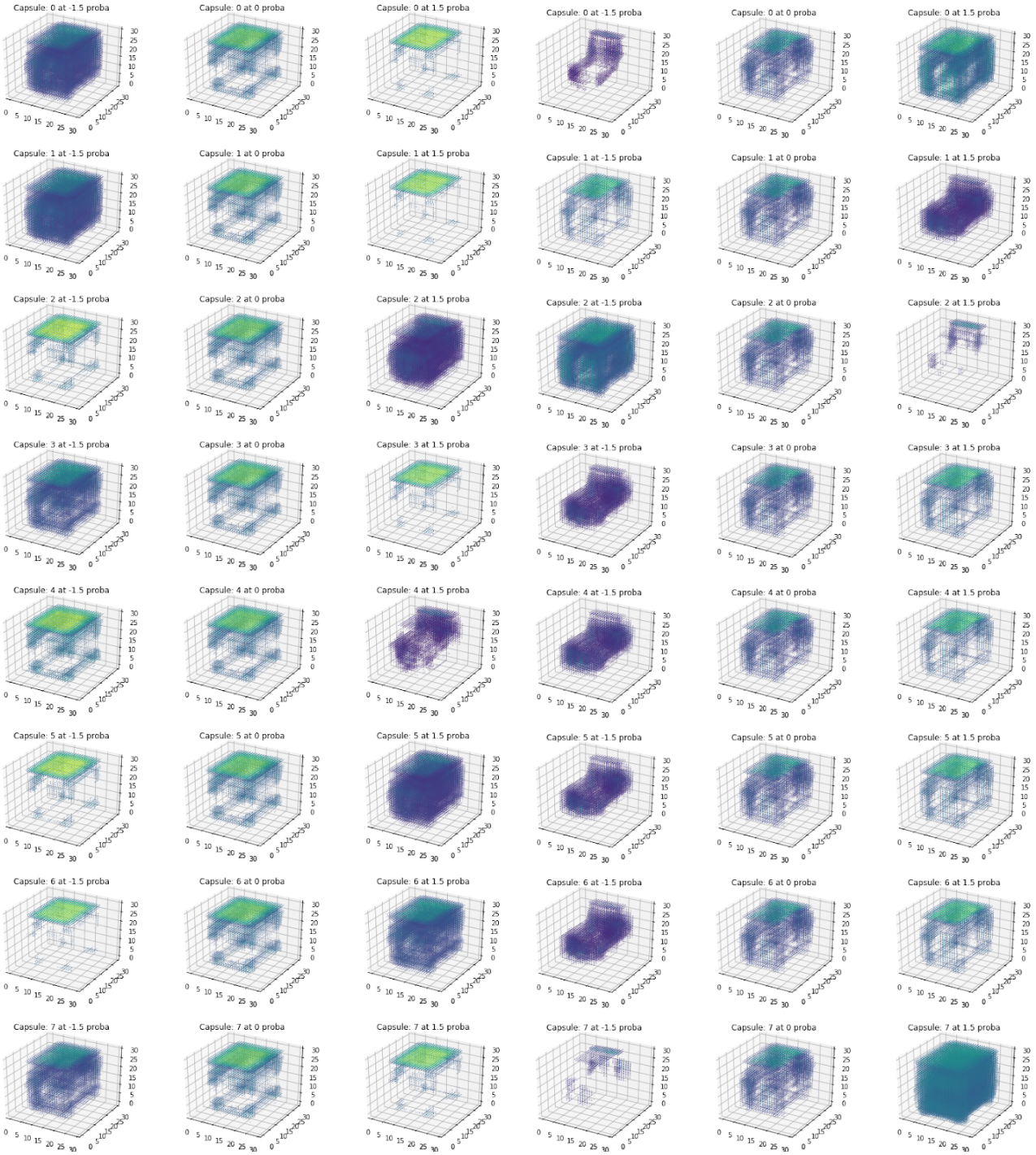


For more examples see the file `3d_model_retriever/EDA_2-Copy1.ipynb

Night Stand True Positive and True Negative (dresser True Positive) High Discriminative Power

True Positive Night Stand
Confidence > .93

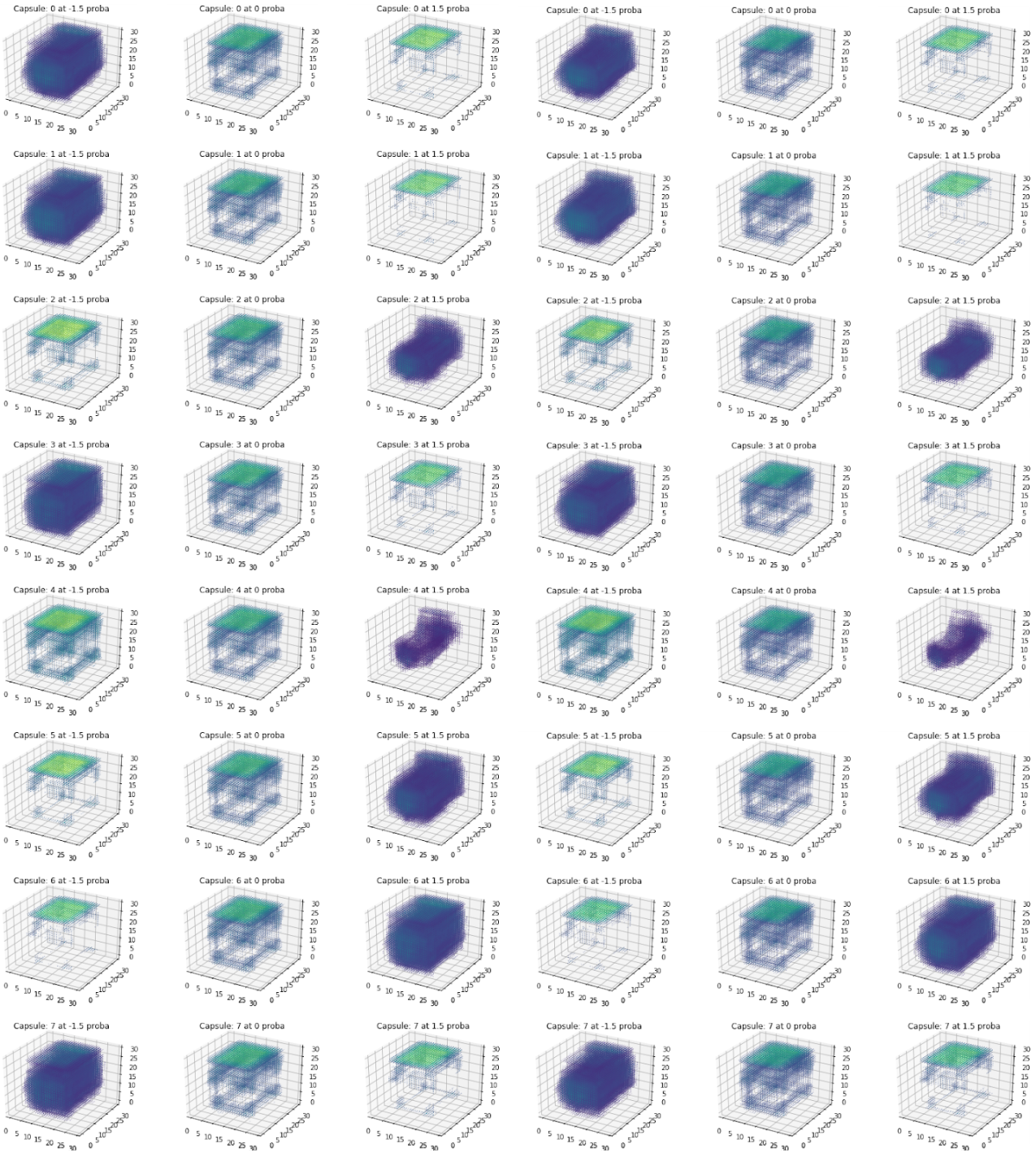
True Positive Dresser
Confidence > .89



Night Stand TP vs FN

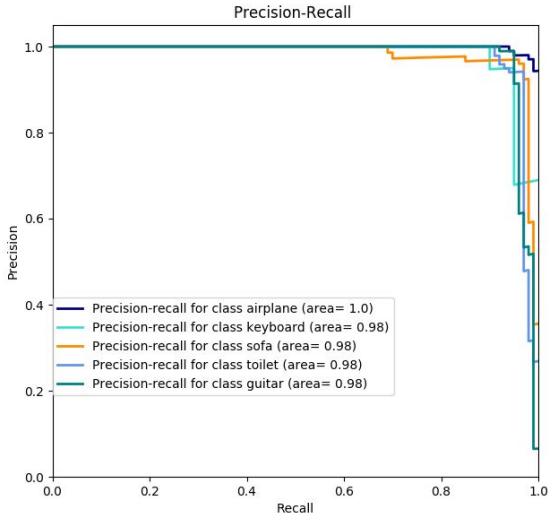
True Positive

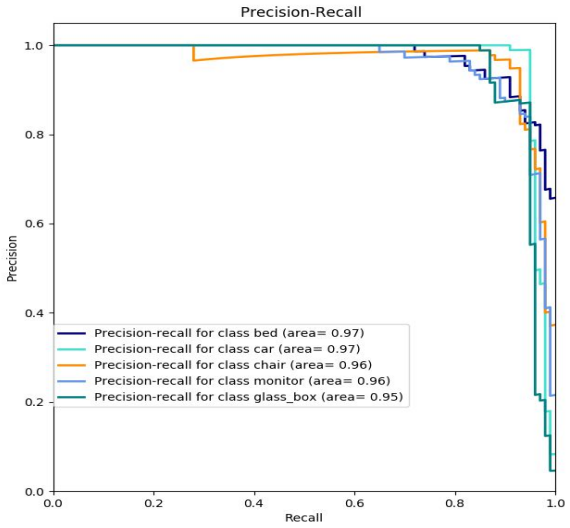
False Negative



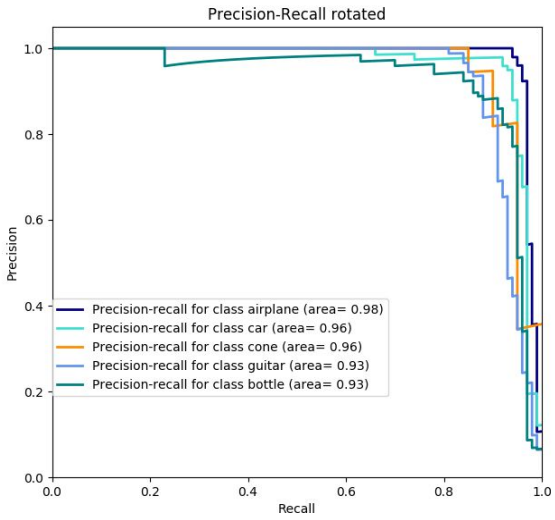
Rotation Invariance ModelNet40

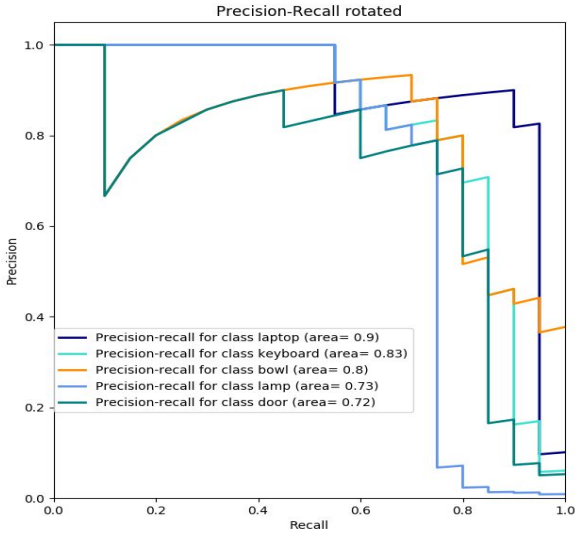
Not Rotated

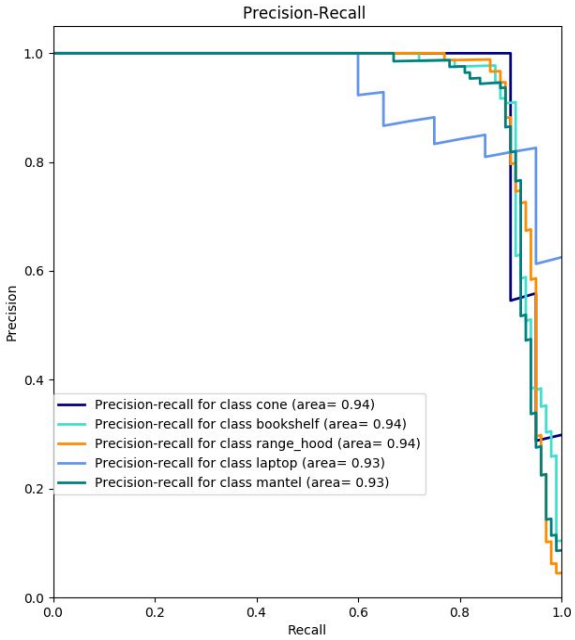


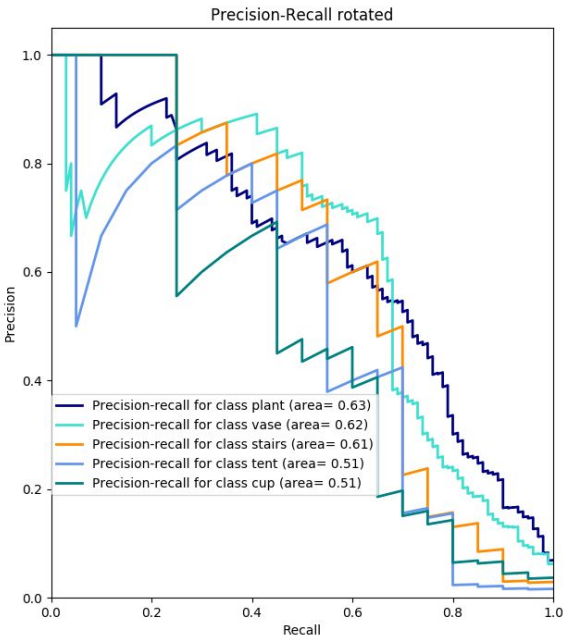


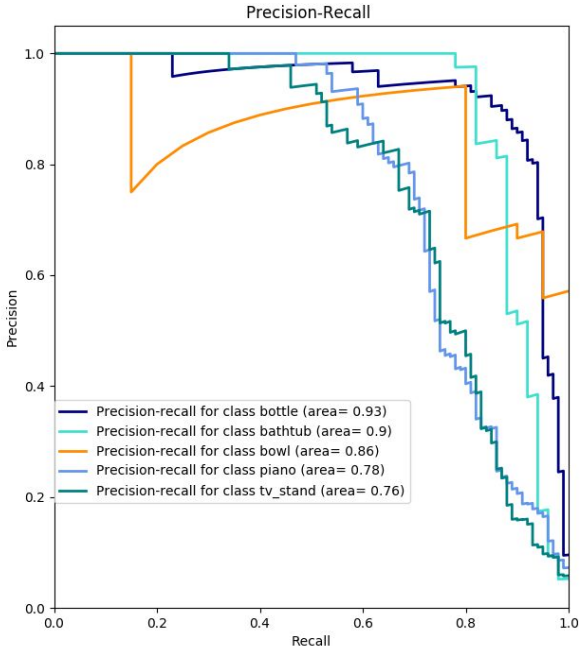
Rotated

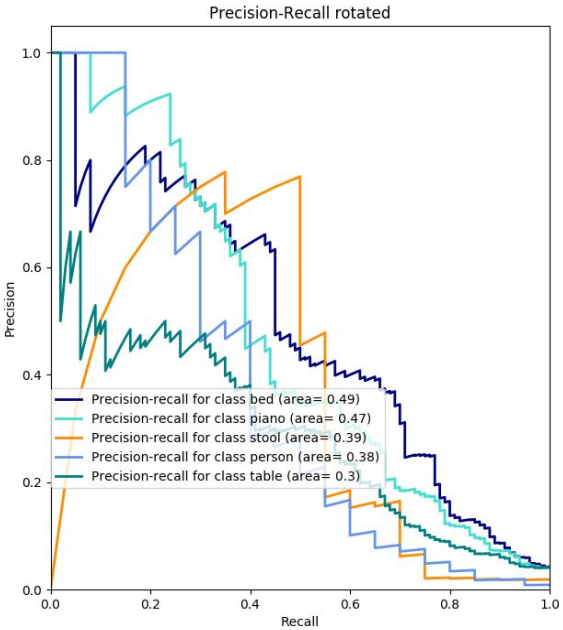


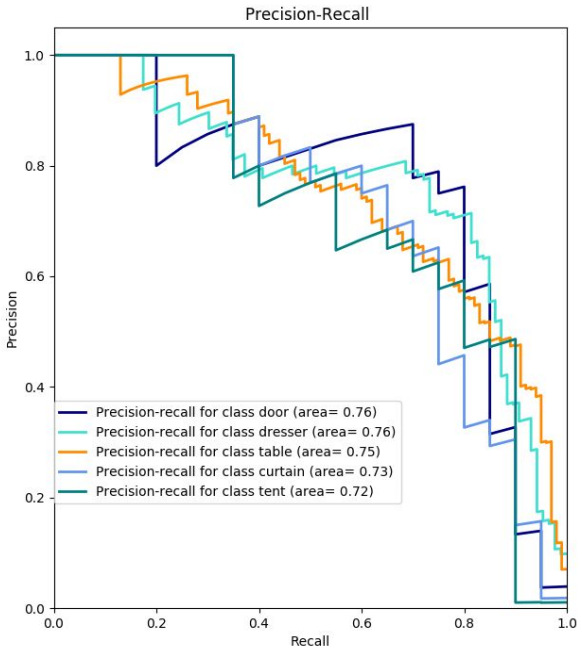


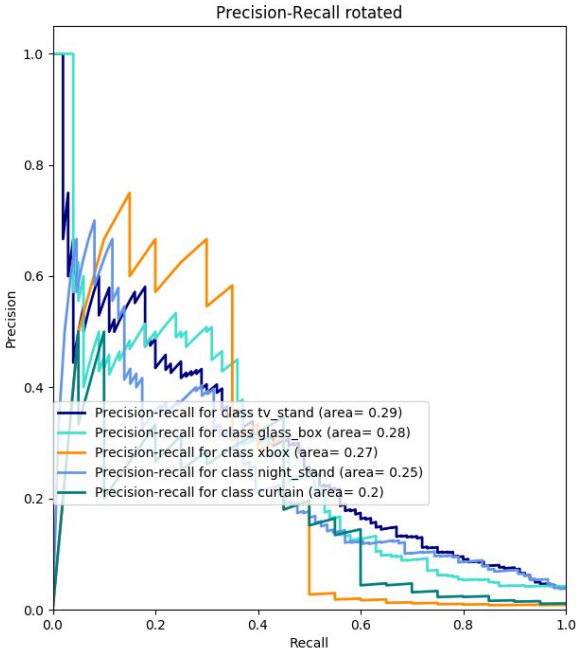

Precision-Recall
Precision
Recall
Precision-recall for class cone (area= 0.94)
Precision-recall for class bookshelf (area= 0.94)
Precision-recall for class range_hood (area= 0.94)
Precision-recall for class laptop (area= 0.93)
Precision-recall for class mantel (area= 0.93)

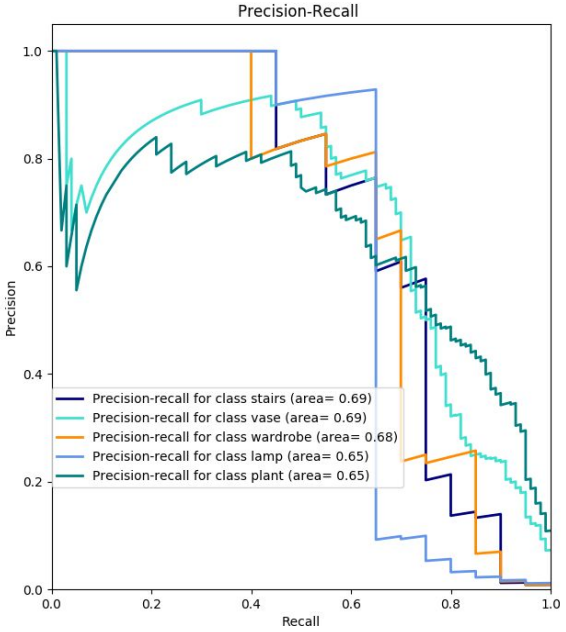

Precision-Recall rotated
Precision
Recall
Precision-recall for class plant (area= 0.63)
Precision-recall for class vase (area= 0.62)
Precision-recall for class stairs (area= 0.61)
Precision-recall for class tent (area= 0.51)
Precision-recall for class cup (area= 0.51)

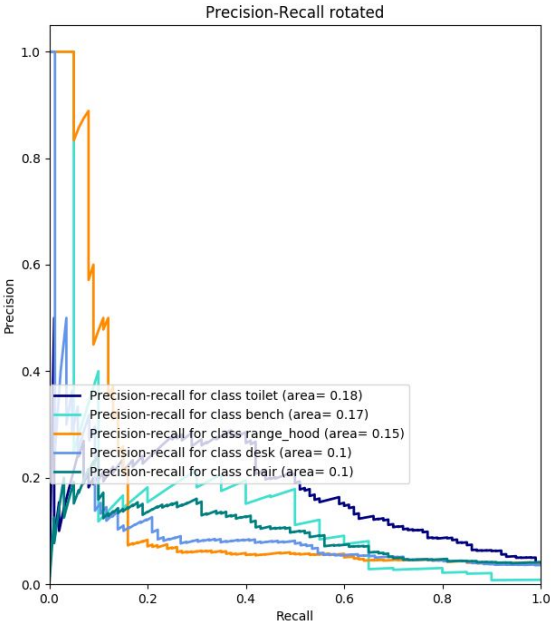

Precision-Recall
Precision
Recall
Precision-recall for class bottle (area= 0.93)
Precision-recall for class bathtub (area= 0.9)
Precision-recall for class bowl (area= 0.86)
Precision-recall for class piano (area= 0.78)
Precision-recall for class tv_stand (area= 0.76)

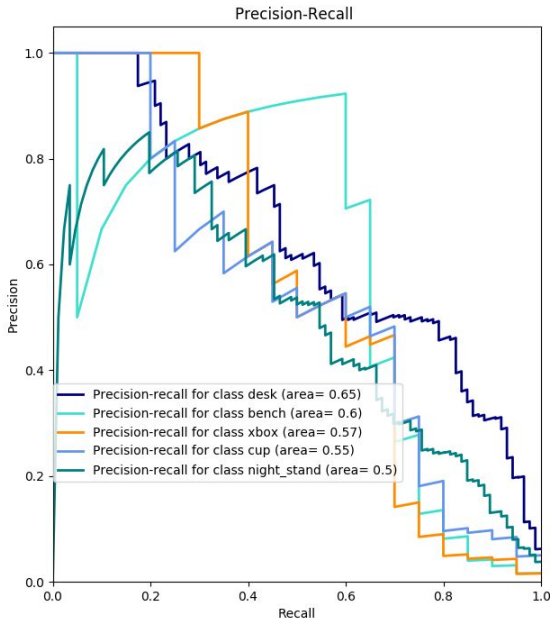

Precision-Recall rotated
Precision
Recall
Precision-recall for class bed (area= 0.49)
Precision-recall for class piano (area= 0.47)
Precision-recall for class stool (area= 0.39)
Precision-recall for class person (area= 0.38)
Precision-recall for class table (area= 0.3)

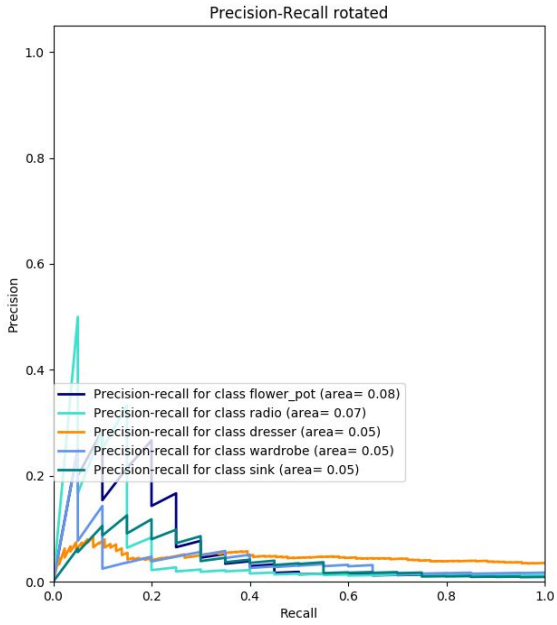

Precision-Recall
Precision
Recall
Precision-recall for class door (area= 0.76)
Precision-recall for class dresser (area= 0.76)
Precision-recall for class table (area= 0.75)
Precision-recall for class curtain (area= 0.73)
Precision-recall for class tent (area= 0.72)

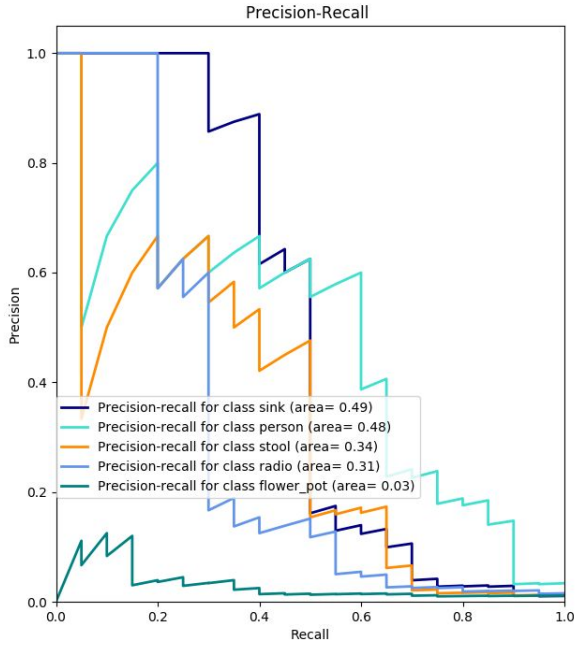

Precision-Recall rotated
Precision
Recall
Precision-recall for class tv_stand (area= 0.29)
Precision-recall for class glass_box (area= 0.28)
Precision-recall for class xbox (area= 0.27)
Precision-recall for class night_stand (area= 0.25)
Precision-recall for class curtain (area= 0.2)

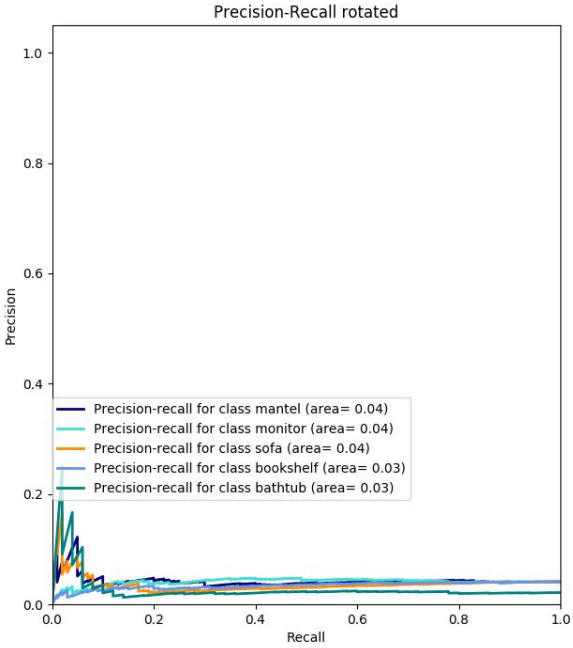

Precision-Recall
Precision
Recall
Precision-recall for class stairs (area= 0.69)
Precision-recall for class vase (area= 0.69)
Precision-recall for class wardrobe (area= 0.68)
Precision-recall for class lamp (area= 0.65)
Precision-recall for class plant (area= 0.65)

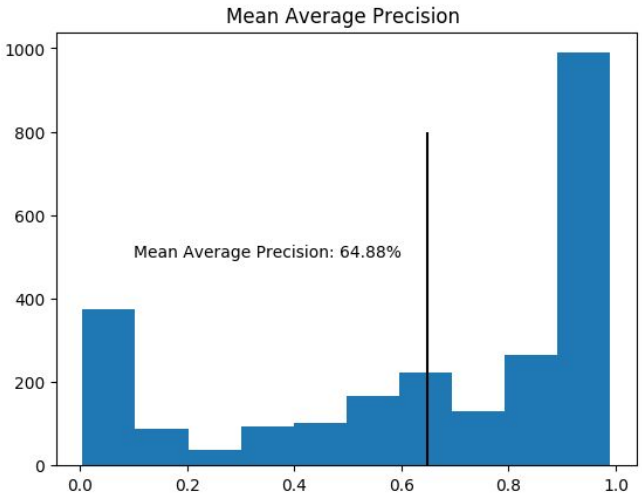

Precision-Recall rotated
Precision
Recall
Precision-recall for class toilet (area= 0.18)
Precision-recall for class bench (area= 0.17)
Precision-recall for class range_hood (area= 0.15)
Precision-recall for class desk (area= 0.1)
Precision-recall for class chair (area= 0.1)

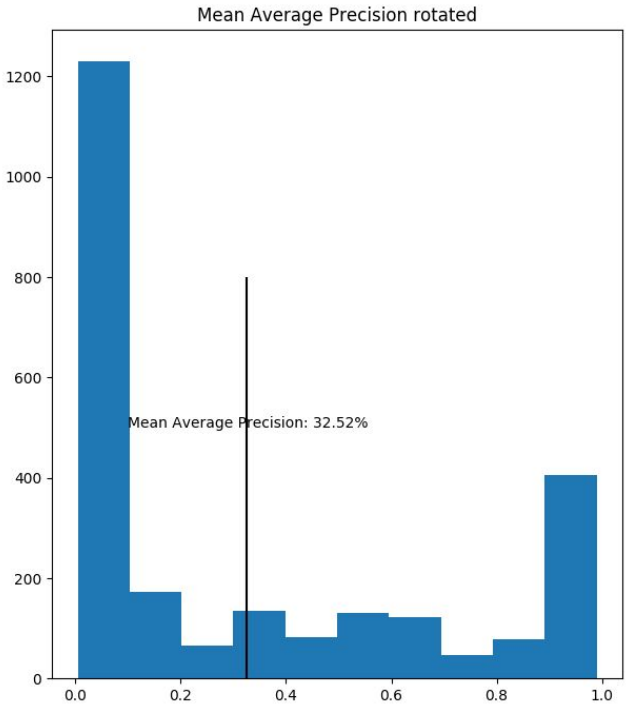

Precision-Recall
Precision
Recall
Precision-recall for class desk (area= 0.65)
Precision-recall for class bench (area= 0.6)
Precision-recall for class xbox (area= 0.57)
Precision-recall for class cup (area= 0.55)
Precision-recall for class night_stand (area= 0.5)


Precision-Recall rotated
Precision
Recall
Precision-recall for class flower_pot (area= 0.08)
Precision-recall for class radio (area= 0.07)
Precision-recall for class dresser (area= 0.05)
Precision-recall for class wardrobe (area= 0.05)
Precision-recall for class sink (area= 0.05)

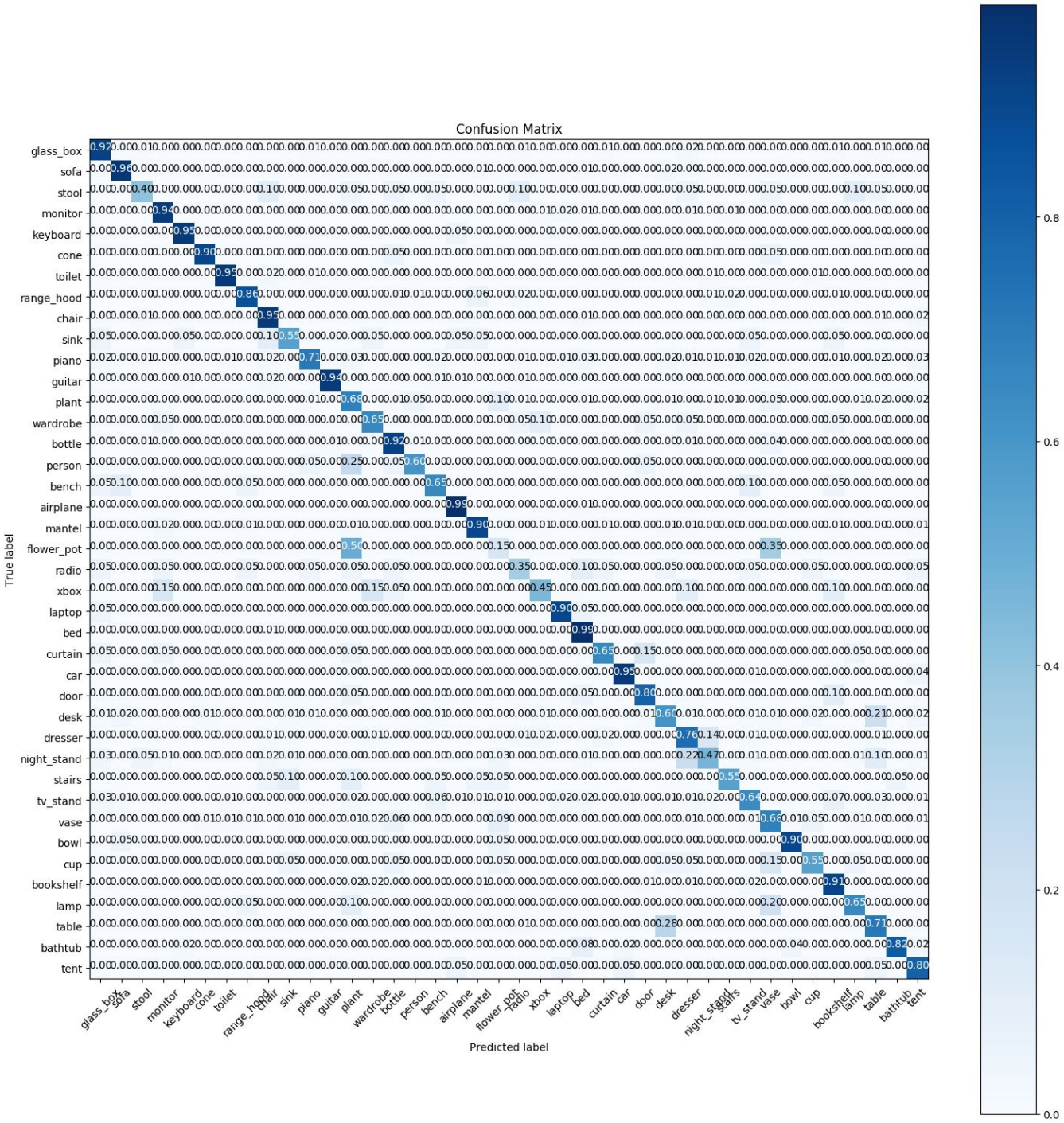

Precision-Recall
Precision
Recall
Precision-recall for class sink (area= 0.49)
Precision-recall for class person (area= 0.48)
Precision-recall for class stool (area= 0.34)
Precision-recall for class radio (area= 0.31)
Precision-recall for class flower_pot (area= 0.03)


Precision-Recall rotated
Precision
Recall
Precision-recall for class mantel (area= 0.04)
Precision-recall for class monitor (area= 0.04)
Precision-recall for class sofa (area= 0.04)
Precision-recall for class bookshelf (area= 0.03)
Precision-recall for class bathtub (area= 0.03)

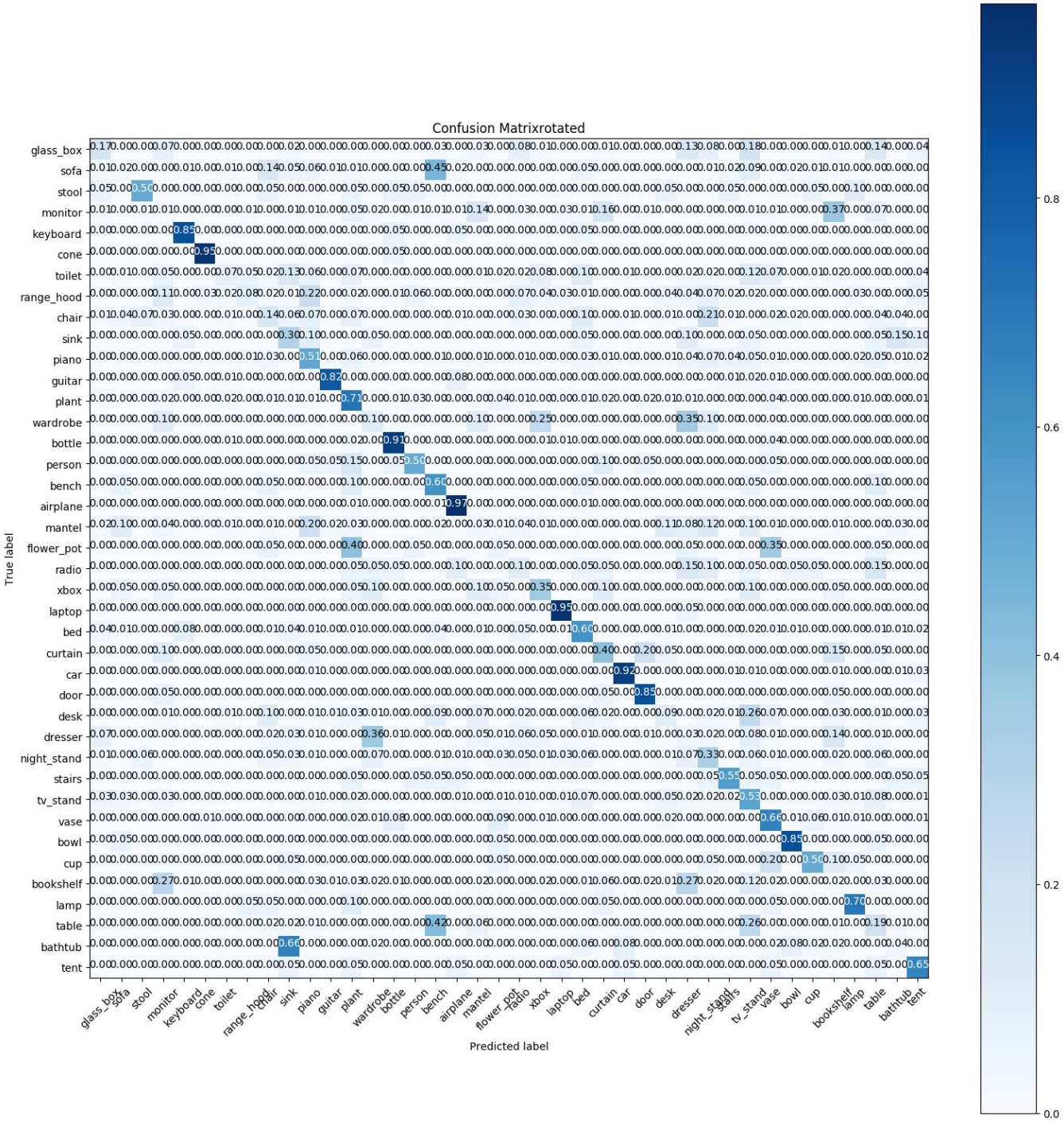

Mean Average Precision
Mean Average Precision: 64.88%
0
200
400
600
800
1000
0.0
0.2
0.4
0.6
0.8
1.0


Mean Average Precision rotated
Mean Average Precision: 32.52%
0
200
400
600
800
1000
1200
0.0
0.2
0.4
0.6
0.8
1.0

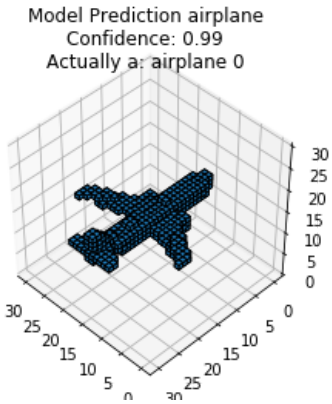
ModelNet40 Confusion Matrix

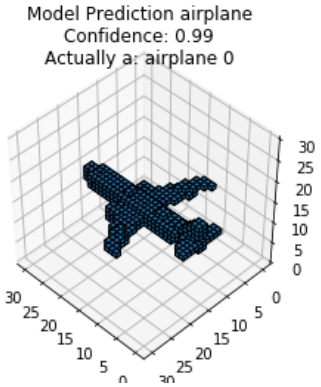


ModelNet40 Confusion Matrix Rotated



Airplane





1 **2** **3** **1** **2** **3**

Manual Random Restart and Warm Up

From 3d_model_retriever/modelnet40_arch.py:138-147

```
    train_model.fit([x_train, y_train], [y_train, x_train],
                                    batch_size=32, epochs=3,
                                    validation_data=[[x_val, y_val], [y_val, x_val]])
                    #                 callbacks=[tb, checkpointer])
                                    # callbacks=[tb, csv, reduce_lr, early_stop])
    train_model.fit([x_train, y_train], [y_train, x_train],
                                    batch_size=128, epochs=NUM_EPOCHS,
                                    validation_data=[[x_val, y_val], [y_val, x_val]],
                    #                 callbacks=[tb, checkpointer])
                                    callbacks=[tb, csv, reduce_lr, early_stop])
```

Benchmark

Algorithm	ModelNet40 Classification (Accuracy)	ModelNet40 Retrieval (MAP)	ModelNet10 Classification (Accuracy)	ModelNet10 Retrieval (MAP)
Minto et al.[33]	89.3%		93.6%	
RotationNet[32]	97.37%		98.46%	
LonchaNet[31]			94.37	
Achlioptas et al. [30]	84.5%		95.4%	
PANORAMA-ENN [29]	95.56%	86.34%	96.85%	93.28%
3D-A-Nets [28]	90.5%	80.1%		
Soltani et al. [27]	82.10%			
Arvind et al. [26]	86.50%			
LonchaNet [25]			94.37%	
3DmFV-Net [24]	91.6%		95.2%	
Zanuttigh and Minto [23]	87.8%		91.5%	
Wang et al. [22]	93.8%			
ECC [21]	83.2%		90.0%	
PANORAMA-NN [20]	90.7%	83.5%	91.1%	87.4%
MVCNN-MultiRes [19]	91.4%			
FPNN [18]	88.4%			
PointNet[17]	89.2%			
Klokov and Lempitsky[16]	91.8%		94.0%	
LightNet[15]	88.93%		93.94%	
Xu and Todorovic[14]	81.26%		88.00%	
Geometry Image [13]	83.9%	51.3%	88.4%	74.9%
Set-convolution [11]	90%			
PointNet [12]			77.6%	
3D-GAN [10]	83.3%		91.0%	
VRN Ensemble [9]	95.54%		97.14%	
ORION [8]			93.8%	
FusionNet [7]	90.8%		93.11%	
Pairwise [6]	90.7%		92.8%	
MVCNN [3]	90.1%	79.5%		
GIFT [5]	83.10%	81.94%	92.35%	91.12%
VoxNet [2]	83%		92%	
DeepPano [4]	77.63%	76.81%	85.45%	84.18%
3DShapeNets [1]	77%	49.2%	83.5%	68.3%

Grid Search ModelNet10

5-Fold Stratified Shuffled Cross validation set =

accuracy	mean_avg_prec	rot_accuracy	rot_mean_avg_prec	primary_cap_kernel_size	n_channels	lam_recon	first_layer_kernel_size	dim_sub_capsule	dim_primary_capsule	conv_layer_filters	NUM_EPOCHS
0.93598	0.88616	0.31707	0.35415	9	1	0.04	9	8	8	256	10
0.93463	0.89393	0.33606	0.37493	9	1	0.04	9	8	8	256	10
0.93463	0.88924	0.27988	0.33032	9	1	0.04	9	8	8	256	10
0.92449	0.87220	0.31122	0.33403	9	1	0.04	9	8	8	256	10
0.92426	0.88023	0.32139	0.36149	9	1	0.04	9	8	8	256	10
0.90969	0.84778	0.34251	0.37395	9	1	0.04	9	8	8	256	10
0.90198	0.83673	0.32930	0.35308	9	2	0.04	9	16	8	128	10
0.89317	0.82704	0.32709	0.34672	9	3	0.04	9	16	4	128	10
0.89207	0.82522	0.33260	0.35360	9	2	0.04	9	16	4	128	10
0.88656	0.82756	NaN	NaN	9	4	0.04	9	8	8	48	10
0.88436	0.82462	NaN	NaN	9	4	0.04	9	8	4	48	10
0.87996	0.81304	NaN	NaN	9	4	0.04	9	16	8	48	10
0.87885	0.81921	NaN	NaN	9	4	0.04	9	8	8	24	10
0.87775	0.80771	NaN	NaN	9	4	0.04	9	16	8	24	10
0.87775	0.80390	NaN	NaN	7	4	0.04	9	16	8	24	10
0.87445	0.81544	NaN	NaN	9	4	0.04	9	16	4	24	10
0.87225	0.80079	0.31938	0.32860	9	3	0.04	9	16	8	128	10
0.87115	0.81003	NaN	NaN	9	4	0.04	9	16	4	48	10
0.85683	0.79138	NaN	NaN	9	4	0.04	9	8	4	24	10
0.85242	0.76423	NaN	NaN	7	4	0.04	9	16	4	48	10

0.85022	0.75254	NaN	NaN	7	4	0.04	9	8	8	24	10
0.85022	0.73431	NaN	NaN	7	4	0.04	9	8	8	48	10
0.83921	0.75976	NaN	NaN	9	4	0.04	9	16	8	48	2
0.83480	0.74217	NaN	NaN	7	4	0.04	9	8	4	48	10
0.83040	0.72242	NaN	NaN	7	4	0.04	9	16	4	24	10
0.72467	0.59002	NaN	NaN	7	4	0.04	9	8	4	24	10
0.11013	0.19332	NaN	NaN	7	4	0.04	9	16	8	48	10
0.09471	0.19332	0.94710	0.19332	9	32	0.04	9	16	8	256	10

Last Row is out of box score. It’s quite low.

Grid Search ModelNet40

5-Fold Stratified Shuffled Cross validation set =

accuracy	mean_avg_prec	rot_accuracy	rot_mean_avg_prec	primary_cap_kernel_size	n_channels	lam_recon	first_layer_kernel_size	dim_sub_capsule	dim_primary_capsule	conv_layer_filters	NUM_EPOCHS
0.84043	0.71472	0.42512	0.31389	9	1	0.04	9	8	8	256	10
0.82975	0.70212	0.41021	0.30473	9	1	0.04	9	8	8	256	10
0.82852	0.70405	0.43803	0.31818	9	1	0.04	9	8	8	256	10
0.82700	0.70031	0.39410	0.29524	9	1	0.04	9	8	8	256	10
0.81092	0.68386	0.38712	0.29252	9	1	0.04	9	8	8	256	10
0.80511	0.64885	0.39384	0.32519	9	1	0.04	9	8	8	256	50
0.76297	0.59170	NaN	NaN	9	4	0.04	9	16	4	24	10
0.75486	0.58689	NaN	NaN	9	8	0.04	9	16	4	24	10
0.73744	0.55776	NaN	NaN	9	4	0.04	9	8	4	24	10
0.73622	0.54835	NaN	NaN	9	8	0.04	9	8	4	24	10
0.68882	0.48273	NaN	NaN	5	4	0.04	9	8	4	24	10
0.61912	0.41612	NaN	NaN	5	4	0.04	9	16	4	24	10
0.07000	0.13858	NaN	NaN	5	5	0.04	9	16	5	48	2