CV Project Report (Spring2020)

A Point Set Generation Network for 3D Object Reconstruction from a Single Image (CVPR December 2016)

Paper by:

Haoqiang Fan*
Hao Su*
Leonidas Guibas

Implemented by:

Team: Vicissitude (30)
Apoorva Srivastava (2019702014)
Anurag Sahu (2018121004)
Kajal Sanklecha (2019801006)
Guide: Dr. Avinash Sharma
TA: Aditya Aggarwal

Contents

Introduction

The current paper deals with the problem of 3D reconstruction from a single 2D image.Multiple techniques have been applied to achieve the same using Deep Learning and the output 3D are always either a collection of images or depth maps,or volumetric representations etc.However, these representations obscure the natural invariance of 3D shapes under geometric transformations and also suffer from a number of other issues like such representations lead to difficult trade offs between sampling resolution and net efficiency. Furthermore, they enshrine quantization artifacts that obscure natural invariances of the data under rigid motions etc.

In this paper,for the first time it has been tried to obtain the output as a collection of unordered 3D point coordinates.

There are following advantages of obtaining output in this form :

1. A point cloud is a simple, uniform structure that is easier to learn, as it does not have to encode multiple primitives or combinatorial connectivity patterns.
2. A point cloud allows simple manipulation when it comes to geometric transformations and deformations, as connectivity does not exist.
3. One advantage of point sets comes from its un-orderedness. Unlike 2D based representations like the depth map no topological constraint is put on the represented object.

But if point clouds are this much advantageous then the question arises why do the majority of 3D representations are either volumetric or collection of images?

It is because extant deep net architectures for both discriminative and generative learning in the signal domain are well suited to data that is regularly sampled, such as images, audio, or video. However, most common 3D geometry representations, such as 2D meshes or point clouds are not regular structures and do not easily fit into architectures that exploit such regularity for weight sharing, etc. That is why the majority of extant works on using deep nets for 3D data resort to either volumetric grids or collections of images.

Challenges

This paper addresses the problem of 3D reconstruction from a single image, generating a straight-forward form of output – point cloud coordinates. Along with this problem arises a unique and interesting issue, that the groundtruth shape for an input image may be ambiguous.As Neural Network has to hallucinate complete 3D structure from a single view, the problem is under-determined hence opening the possibility of multiple possible outputs for a single 2D image.The problem is quite different from general regression problem where there is proper annotation for the training data and the output is deterministic.Here the ground truth is one possible 3D output out of all others equally probable on which net can be trained.
Due to ground truth ambiguity,the network design,the choice of loss function and the learning paradigm all become very crucial and their choice becomes highly sensitive in deciding the correctness of the output.

To deal with the above situation,the authors have chosen the following:

- A Conditional Generative Network
- Different Versions of Network Design
- Loss function of Chamfer Distance(Other than usual norms)
- Introduction of Random Vector ‘r’ & MoN Loss function as a wrapping function to deal with multiple outputs.

The reason for the above choices and their roles in getting the optimal output has been explained in the upcoming sections.

Network Architecture

Why Conditional Generative Network?

To deal with the challenges arose due to the choice of unorthodox output, multiple steps were taken by the authors,out of them one was to choose the architecture of the neural network to be Conditional Generative Network rather than the Discriminative Network.It is because this is a situation where there are multiple, equally good 3D reconstructions of a 2D image, making our problem very different from classical regression/classification settings, where each training sample has a unique ground truth annotation.

The network being generative tries to learn the whole distribution of the input image space and provides the output from the same distribution.The network is conditional sampler so that it samples plausible 3D point clouds from the estimated ground truth space given an input image.

Secondly, authors allow multiple reconstruction candidates for a single input image. This design reflects the fact that a single image cannot fully determine the reconstruction of a 3D shape.

Network Formulation:

Goal **-** Single 2D image (RGB or RGBD) to complete 3D shape

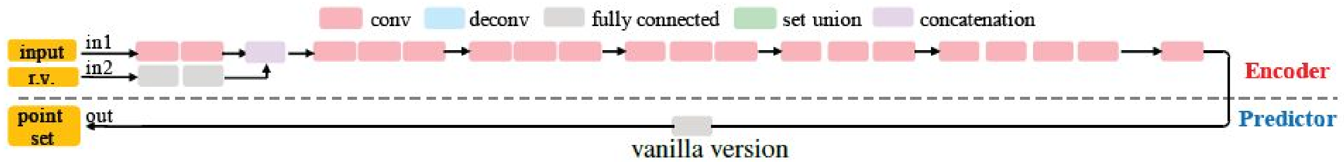
Output Format**-** $S = \{(x_i, y_i, z_i)\}^{N}_{i=1}$; where N is predefined constant, N = 1024

Ground Truth**-**The ground truth is a probability distribution, P(./I) over the shapes possible for input, I i.e. for a given image the output 3D points come with a probability.So, Neural Network is trained as a conditional sampler from P(./I)

$S = G(I, r, \theta)$
Θ - n/w parameter
$r \sim N(O, I)$ - It is a random variable to perturb the input

Basic structure of the network:

Vanilla Version:

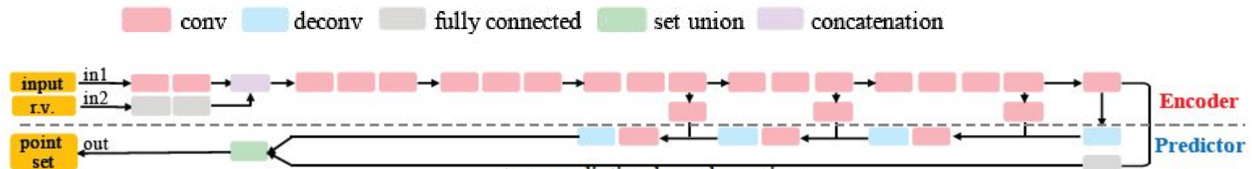


vanilla version

Network has an Encoder Stage and a Predictor Stage.
Encoder Stage: The encoder maps the input pair of an image I and a random vector r into an embedding space which can be understood by the predictor.The encoder is a composition of convolution and ReLUlayers, in addition, a random vector r is subsumed so that it perturbs the prediction from the image I.
Predictor Stage: The predictor outputs a shape as an N x 3 matrix M, each row containing the coordinates of one point.It contains one fully connected layer.

Though simple, this version works reasonably well in practice. To further improve the design of the predictor branch to better accommodate large and smooth surfaces which are common in natural objects,a Two prediction branch version was designed.

Two prediction branch version:



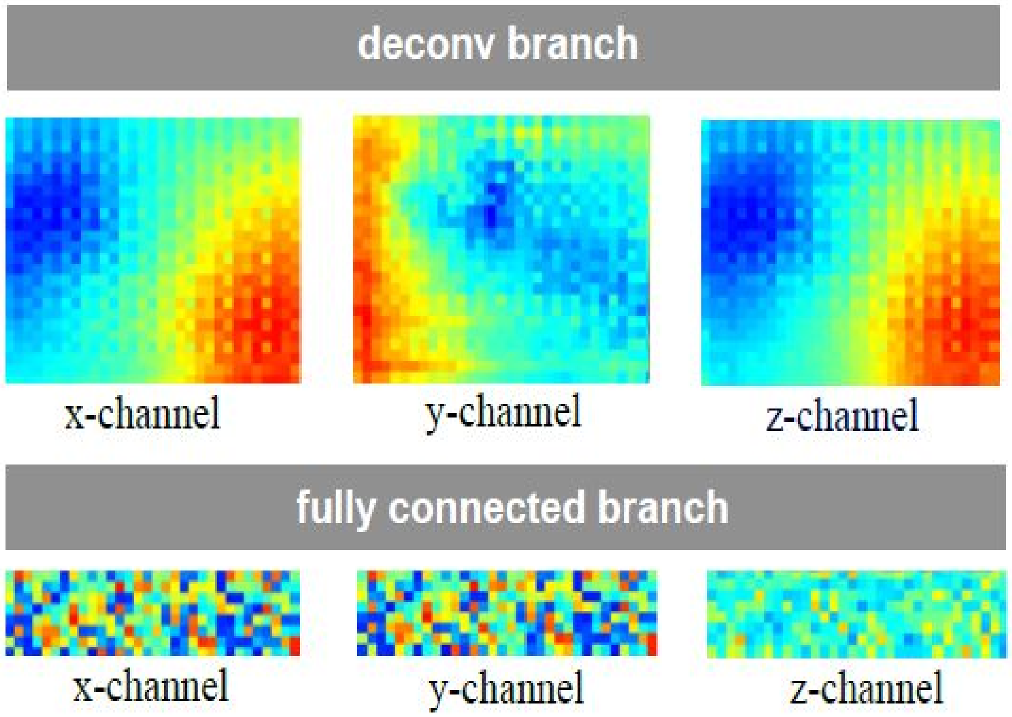
two prediction branch version

This version has two parallel predictor branches – a fully-connected (fc) branch and a deconvolution (deconv) branch. The fc branch predicts N1 points as before. The deconv branch predicts a 3 channel image of size HxW, of which the three values at each pixel are the coordinates of a point, giving another HxW points. Their predictions are later merged together to form the whole set of points in M. Multiple skip links are added to boost information flow across encoder and predictor.

The fully connected predictor as above cannot make full use of such natural geometric statistics, since each point is predicted independently. The(conv-deconv) predictor exploits this geometric smoothness property.
Due to which one enjoys high flexibility in capturing complicated structures(fully connected)and the other exploits geometric continuity(conv-deconv).

In summary, with the fc branch, our model enjoys high flexibility,showing good performance at describing intricate structures.
With the deconvolution branch, our model becomes not only more parameter parsimonious by weight sharing;but also more friendly to large smooth surfaces, due to the spatial continuity induced by deconv and conv.

Comparison of fully connected branch and deconv branch:



In above Fig the values in the x, y and z channels are plotted as 2D images for one of the models. In the deconv branch the network learns to use the convolution structure to construct a 2D surface that wraps around the object. In the fully connected branch the output is less organized as the channels are not ordered.
The deconv branch is in general good at capturing the "main body" of the object, while the fully connected branch complements the shape with more detailed

components. This reveals the complementarity of the two branches. The predefined weights sharing and node connectivity endow the deconv branch with higher efficiency when they are congruent with the desired output's structure. The fully connected branch is more flexible but the independent control of each point consumes more network capacity.

Loss Function

A critical challenge is to design a good loss function for comparing the predicted point cloud and the groundtruth.

To plug in a neural network, a suitable distance must satisfy at least three conditions:

1) Differentiable with respect to point locations.

2) Efficient to compute, as data will be forwarded and back-propagated multiple times.

3) Robust against a small number of outlier points in the sets.

We seek for a distance d between subsets in R^3, so that the loss function takes the form:

$$L(\{S_i^{pred}\}, \{S_i^{gt}\}) = \sum d(S_i^{pred}, S_i^{gt}),$$

<u>Loss Function Definition:</u>

Chamfer distance We define the Chamfer distance between $S_1, S_2 \subseteq \mathbb{R}^3$ as:

$$d_{CD}(S_1, S_2) = \sum_{x \in S_1} \min_{y \in S_2} \|x - y\|_2^2 + \sum_{y \in S_2} \min_{x \in S_1} \|x - y\|_2^2$$

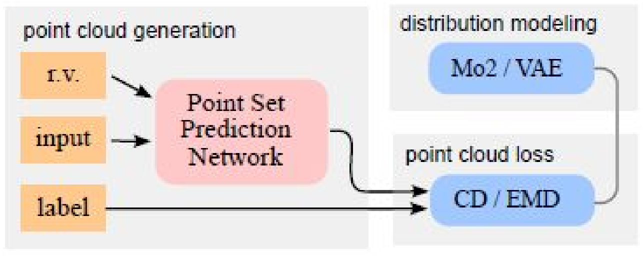
For each point, the algorithm of Chamfer Distance finds the nearest neighbor in the other set and sums the squared distances up. Viewed as a function of point locations in S1 and S2, Chamfer Distance is continuous and piecewise smooth. The range search for each point is independent, thus trivially parallelizable. Also, spatial data structures like KD-tree can be used to accelerate nearest neighbor search. Though simple, Chamfer Distance produces reasonably high quality results in practice.

<u>Need of random vector 'r':</u>
Facing the inherent inability to resolve the shape precisely, neural networks tend to predict a "mean" shape averaging out the space of uncertainty.
As shown in the Paper that mean shape kills the purpose of predicting the multiple shapes and it varies with the choice of the distance.
The ambiguity of groundtruth shape may significantly affect the trained predictor, as the loss function induces our model to predict the mean of the possible shapes.
To better model the uncertainty or inherent ambiguity (e.g. unseen parts in the single view), we enable the system to generate distributional output.
Naively plugging G (I, r, θ) into Loss Function to predict S_{predi} won't work, as the loss minimization will nullify the randomness. It is also unclear how to make CGAN work in our scenario, as building a discriminator that directly consumes a point set is itself an open problem.
The problem can be solved by more complex frameworks like VAE, where we can incorporate secondary input channels (e.g. another view). However, we find practically a simple and effective method for uncertainty modeling:

<u>The MoN loss:</u> We train our network by minimizing a loss function as below:

$$\underset{\Theta}{\text{minimize}} \sum_{k} \min_{\substack{r_j \sim \mathbb{N}(\mathbf{0},\mathbf{I}) \\ 1 \le j \le n}} \{d(\mathbb{G}(I_k, r_j; \Theta), S_k^{gt})\}$$

Given an image I_k, G (I, r, θ) makes n predictions by perturbing the input with n random vectors r_j . Intuitively, we expect that one of the predictions will be close to the ground truth S^{gt}_k given by the training data, meaning that the minimum of the n distances between each prediction and the ground truth must be small.
We name this loss as Min-of-N loss (MoN), since it comes from the minimum of n distances. Any of the point set regression networks can be plugged into the meta network incorporating the MoN loss. In practice, we find that setting n = 2 already enables our method to well explore the groundtruth space.



<u>System Architecture</u>

Implementation Details

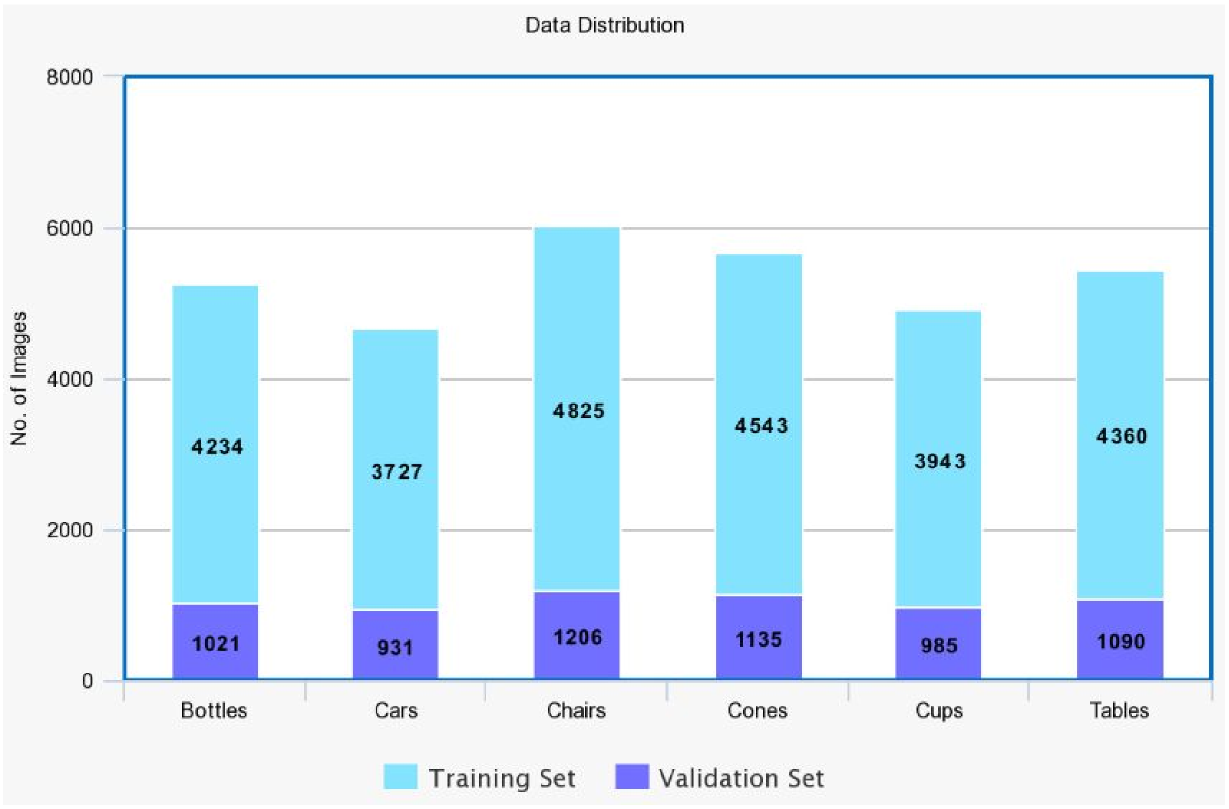
How the dataset is prepared by them:
They rendered 2D views from CAD object models. Models are from the ShapeNet dataset , models with textures, containing large volumes of manually cleaned 3D objects. Concretely they used a subset of 220K models covering 2,000 object categories.
For each model, they normalized the radius of its bounding hemi-sphere to unit 1 and aligned their ground plane.Then each model was rendered into 2D images according to the Blinn-Phong shading formula with randomly chosen environmental maps. They used a simple local lightning model for the sake of computation time. However, it is straight-forward to extend our method to incorporate global illumination algorithms and more complex backgrounds.

Implementation Details:

- Our network works on input images of 192x256.
- The deconv branch produces 768 points, which correspond to a 32x24 three channel image. The fully connected branch produces 256 points. So the final image is 32x32x3,where each location has 3 values depicting x,y,z.
- The training program is implemented in TensorFlow.
- 300000 gradient steps are taken, each computed from a minibatch of 32.
- Adam is used as the optimizer.
- We observed that the training procedure is smooth even without batch normalization.
- All activation functions are ReLU.
- Loss Value: 191.39 for last gradient step possible for Vanilla Version
- Loss Value: 175.38 for last gradient step possible for Two Branch Version
- Regularization: L2 Regularizer
- Number of points in Ground Truth: 16384

Dataset Preparation

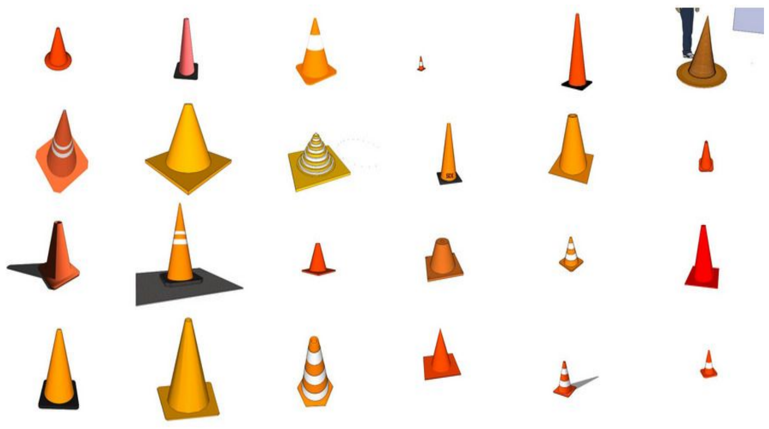
Data has been divided in 6 classes and 20% data of each class is considered to be in validation set as shown below:



Classes:

Cup

Table



Cone

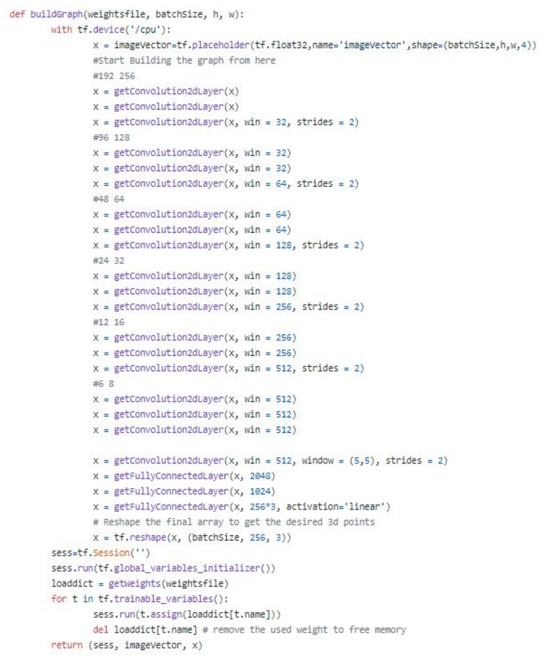
Chair

Bottle

Car

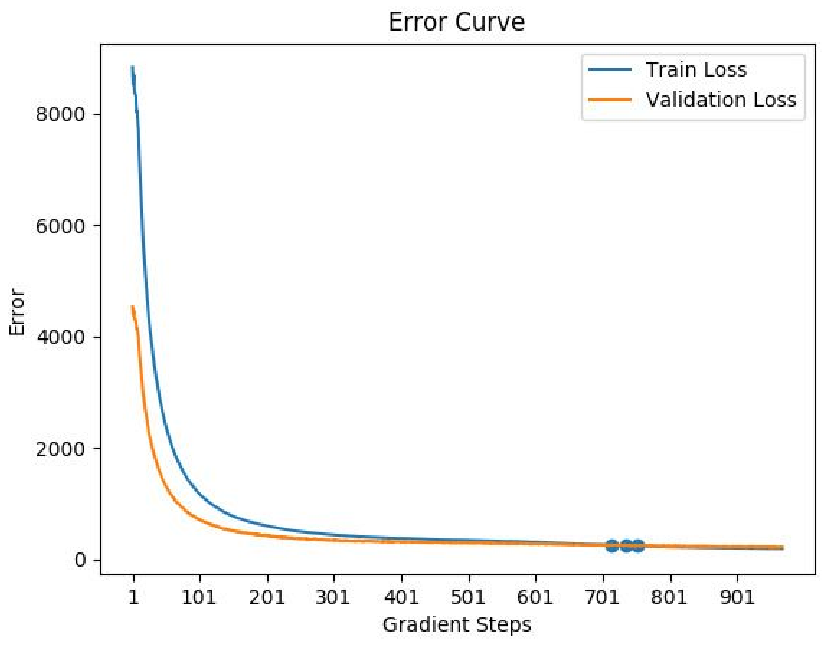
Code and Results

Network Architecture Code for Vanilla Version:

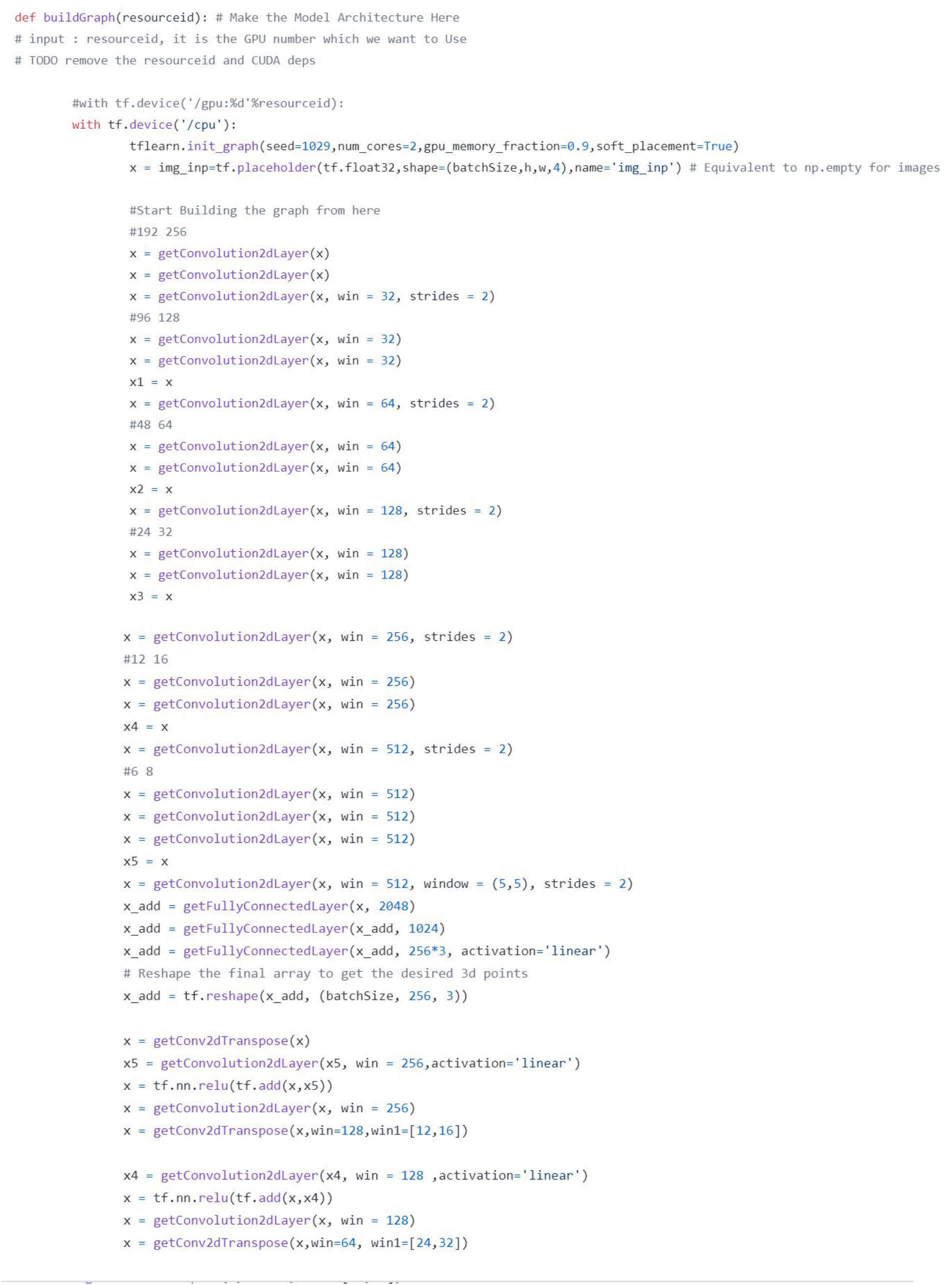


```
def buildGraph(weightsfile, batchSize, h, w):
        with tf.device('/cpu'):
                x = imageVector=tf.placeholder(tf.float32,name='imageVector',shape=(batchSize,h,w,4))
                #Start Building the graph from here
                #192 256
                x = getConvolution2dLayer(x)
                x = getConvolution2dLayer(x)
                x = getConvolution2dLayer(x, win = 32, strides = 2)
                #96 128
                x = getConvolution2dLayer(x, win = 32)
                x = getConvolution2dLayer(x, win = 32)
                x = getConvolution2dLayer(x, win = 64, strides = 2)
                #48 64
                x = getConvolution2dLayer(x, win = 64)
                x = getConvolution2dLayer(x, win = 64)
                x = getConvolution2dLayer(x, win = 128, strides = 2)
                #24 32
                x = getConvolution2dLayer(x, win = 128)
                x = getConvolution2dLayer(x, win = 128)
                x = getConvolution2dLayer(x, win = 256, strides = 2)
                #12 16
                x = getConvolution2dLayer(x, win = 256)
                x = getConvolution2dLayer(x, win = 256)
                x = getConvolution2dLayer(x, win = 512, strides = 2)
                #6 8
                x = getConvolution2dLayer(x, win = 512)
                x = getConvolution2dLayer(x, win = 512)
                x = getConvolution2dLayer(x, win = 512)

                x = getConvolution2dLayer(x, win = 512, window = (5,5), strides = 2)
                x = getFullyConnectedLayer(x, 2048)
                x = getFullyConnectedLayer(x, 1024)
                x = getFullyConnectedLayer(x, 256*3, activation='linear')
                # Reshape the final array to get the desired 3d points
                x = tf.reshape(x, (batchSize, 256, 3))
        sess=tf.Session('')
        sess.run(tf.global_variables_initializer())
        loaddict = getWeights(weightsfile)
        for t in tf.trainable_variables():
                sess.run(t.assign(loaddict[t.name]))
                del loaddict[t.name] # remove the used weight to free memory
        return (sess, imageVector, x)
```

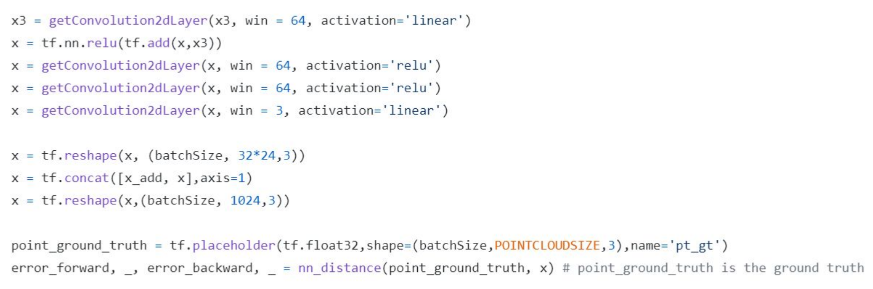
Loss Curve for Vanilla Version:



Network Architecture Code for Two Branch Prediction Version:

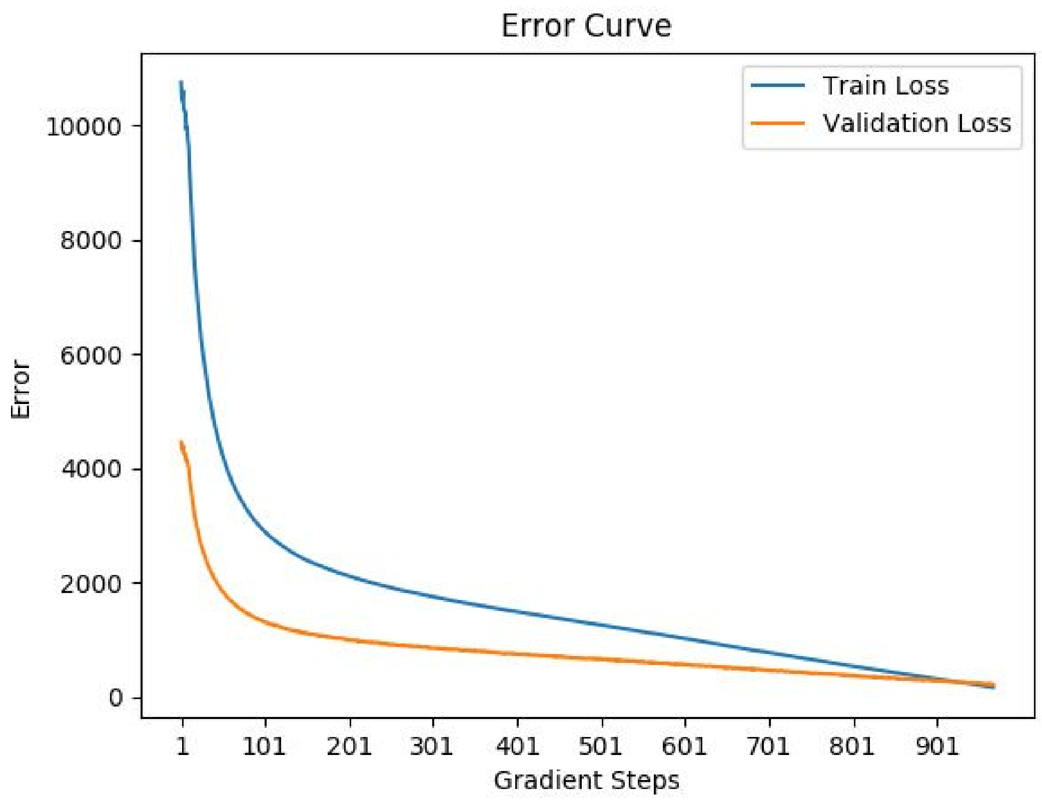


```
def buildGraph(resourceid): # Make the Model Architecture Here
# input : resourceid, it is the GPU number which we want to Use
# TODO remove the resourceid and CUDA deps

        #with tf.device('/gpu:%d'%resourceid):
        with tf.device('/cpu'):
                tflearn.init_graph(seed=1029,num_cores=2,gpu_memory_fraction=0.9,soft_placement=True)
                x = img_inp=tf.placeholder(tf.float32,shape=(batchSize,h,w,4),name='img_inp') # Equivalent to np.empty for images

                #Start Building the graph from here
                #192 256
                x = getConvolution2dLayer(x)
                x = getConvolution2dLayer(x)
                x = getConvolution2dLayer(x, win = 32, strides = 2)
                #96 128
                x = getConvolution2dLayer(x, win = 32)
                x = getConvolution2dLayer(x, win = 32)
                x1 = x
                x = getConvolution2dLayer(x, win = 64, strides = 2)
                #48 64
                x = getConvolution2dLayer(x, win = 64)
                x = getConvolution2dLayer(x, win = 64)
                x2 = x
                x = getConvolution2dLayer(x, win = 128, strides = 2)
                #24 32
                x = getConvolution2dLayer(x, win = 128)
                x = getConvolution2dLayer(x, win = 128)
                x3 = x

                x = getConvolution2dLayer(x, win = 256, strides = 2)
                #12 16
                x = getConvolution2dLayer(x, win = 256)
                x = getConvolution2dLayer(x, win = 256)
                x4 = x
                x = getConvolution2dLayer(x, win = 512, strides = 2)
                #6 8
                x = getConvolution2dLayer(x, win = 512)
                x = getConvolution2dLayer(x, win = 512)
                x = getConvolution2dLayer(x, win = 512)
                x5 = x
                x = getConvolution2dLayer(x, win = 512, window = (5,5), strides = 2)
                x_add = getFullyConnectedLayer(x, 2048)
                x_add = getFullyConnectedLayer(x_add, 1024)
                x_add = getFullyConnectedLayer(x_add, 256*3, activation='linear')
                # Reshape the final array to get the desired 3d points
                x_add = tf.reshape(x_add, (batchSize, 256, 3))

                x = getConv2dTranspose(x)
                x5 = getConvolution2dLayer(x5, win = 256,activation='linear')
                x = tf.nn.relu(tf.add(x,x5))
                x = getConvolution2dLayer(x, win = 256)
                x = getConv2dTranspose(x,win=128,win1=[12,16])

                x4 = getConvolution2dLayer(x4, win = 128 ,activation='linear')
                x = tf.nn.relu(tf.add(x,x4))
                x = getConvolution2dLayer(x, win = 128)
                x = getConv2dTranspose(x,win=64, win1=[24,32])
```

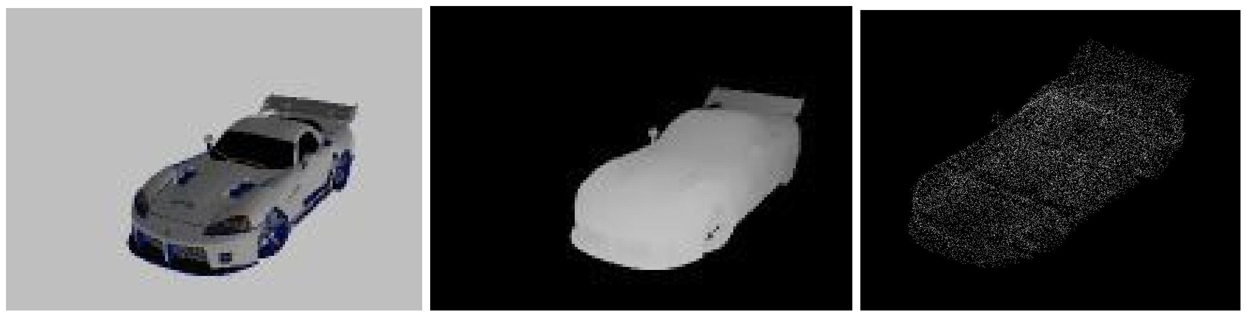


```
        x3 = getConvolution2dLayer(x3, win = 64, activation='linear')
        x = tf.nn.relu(tf.add(x,x3))
        x = getConvolution2dLayer(x, win = 64, activation='relu')
        x = getConvolution2dLayer(x, win = 64, activation='relu')
        x = getConvolution2dLayer(x, win = 3, activation='linear')

        x = tf.reshape(x, (batchSize, 32*24,3))
        x = tf.concat([x_add, x],axis=1)
        x = tf.reshape(x,(batchSize, 1024,3))

        point_ground_truth = tf.placeholder(tf.float32,shape=(batchSize,POINTCLOUDSIZE,3),name='pt_gt')
        error_forward, _, error_backward, _ = nn_distance(point_ground_truth, x) # point_ground_truth is the ground truth
```

Loss Curve for Two Branch Prediction Version:

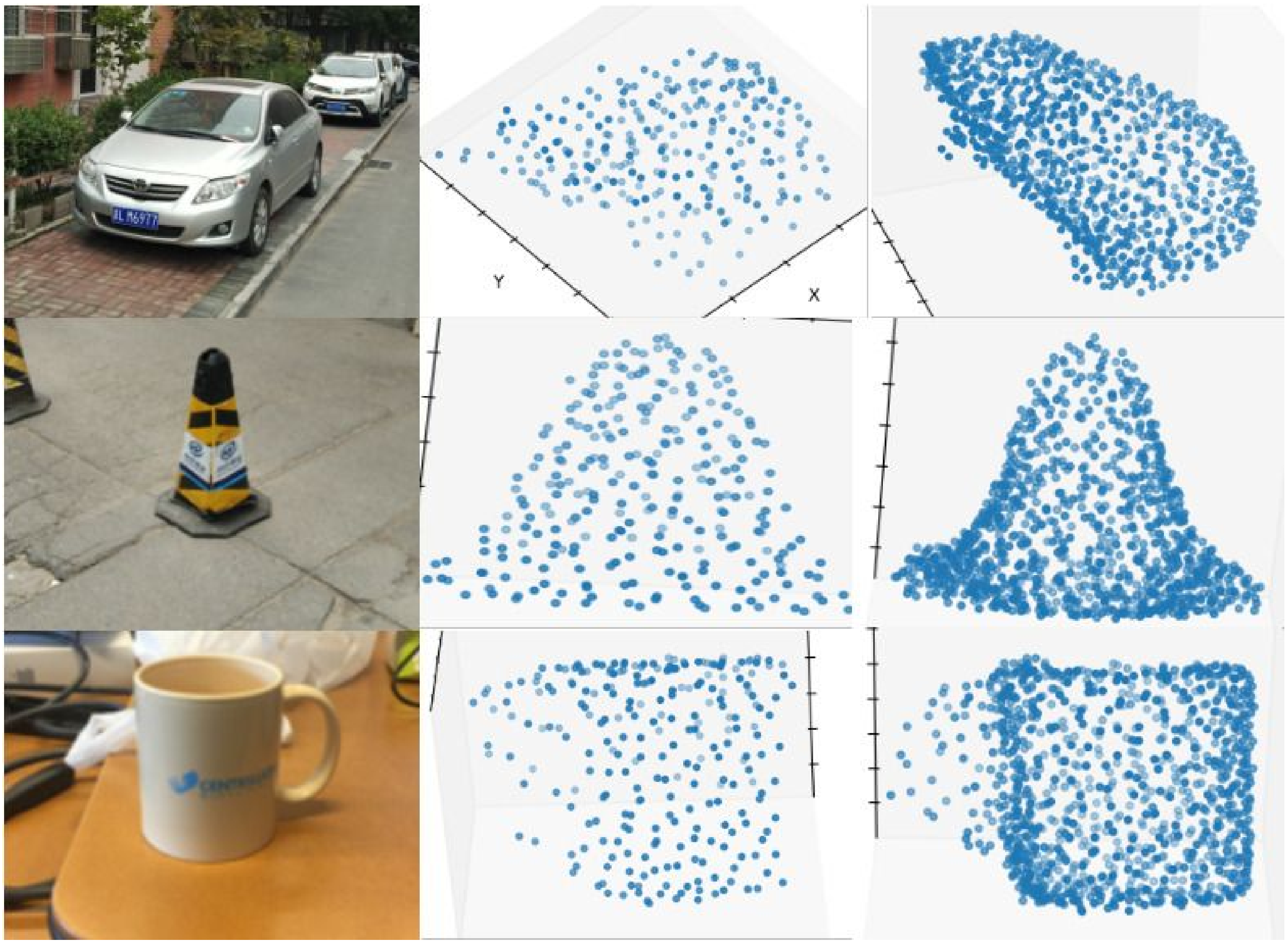


Input Image, Random Vector & Ground Truth:

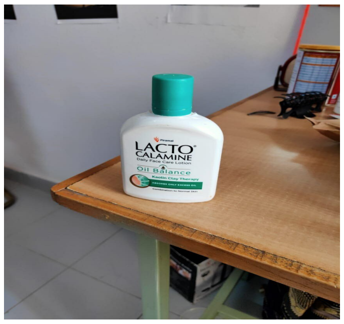
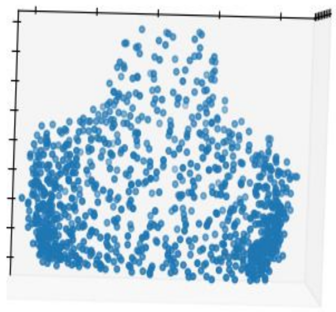
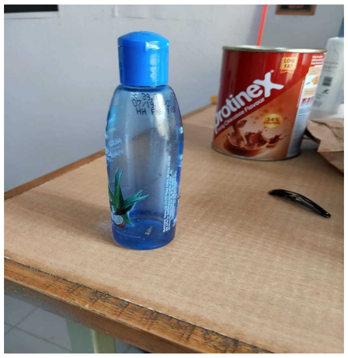
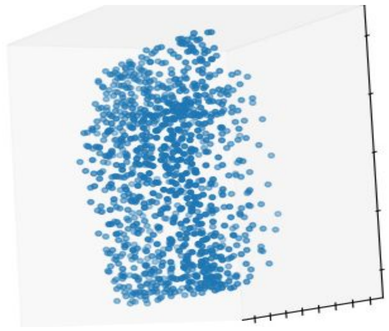
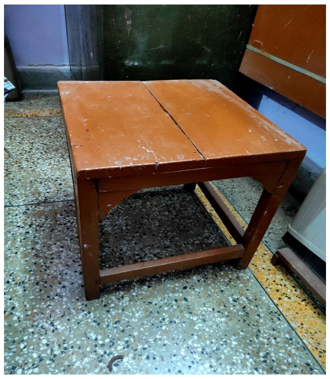
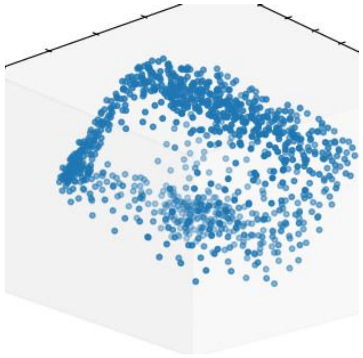
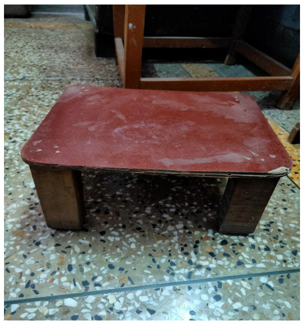
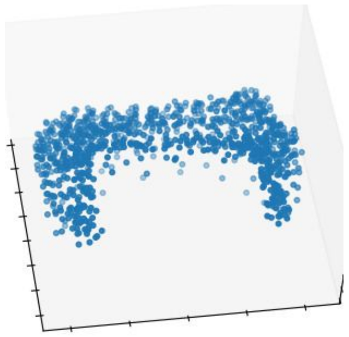


Results on their Data:

Input **Vanilla Version Output** **Two Branch Output**



Results on Real-World Data using 2 prediction version:

Input	Output
	
	
	
	

Conclusion

- Implementing the project gave us a good exposure to Deep Networks
- It helped in Introduction to the Research Area of 3D Reconstruction from monocular image using Deep Neural Networks
- It helped in understanding the steps involved in finding a solution from scratch.
- However,Still much scope remains in terms of improvement of results and understanding the concepts deeper

References

1) http://ai.stanford.edu/~haosu/papers/SI2PC_arxiv_submit.pdf
2) M. Mirza and S. Osindero. Conditional generative adversarial nets.arXiv preprint arXiv:1411.1784,2014.
3) ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], 2015
4) A unified approach for single and multiview 3d object reconstruction. arXiv preprint arXiv:1604.00449, 2016.
5) D. F. Fouhey, A. Gupta, and M. Hebert. Data-driven 3D primitives for single image understanding. In ICCV, 2013.