GraphX-Convolution for Point Cloud Deformation in 2D-to-3D conversion

22-bit image

Aman Rojjha - 2019111018
Tejas Chaudhari - 2019111013

Introduction

3D shape reasoning is important in the field of computer vision as it plays a vital role in robotics, modeling, graphics, and so on. With the current methods, we are able to estimate reliable shapes of the object using multiple images from different viewpoints. But, we humans can reasonably estimate the shape of an object from a single image. Our task is to train machines to do so.
This seems unlikely to be solved since some information is lost when we go from 3D to 2D, however if a machine is able to learn a shape prior like us humans, then it would be able to infer 3D shapes from 2D images reliably.

The key properties of the system which models this behavior are:

- Model should make predictions based on not only local features but also high-level semantics
- Model should consider spatial correlation between points
- Method should be scalable/output point cloud can be of arbitrary size

This approach is better than the original paper we had taken "A Point Set Generation Network for
3D Object Reconstruction from a Single Image, Fan et al" since it is far more scalable. Fan et al had proposed an encoder-decoder architecture with various shortcuts to directly map an input image to its point cloud representation. The disadvantage was that the number of trainable parameters are proportional to the number of points in the output cloud since it directly generates a point cloud. In contrast, in this approach we deform a point cloud instead of generating one, which makes the system far more scalable.

Approach

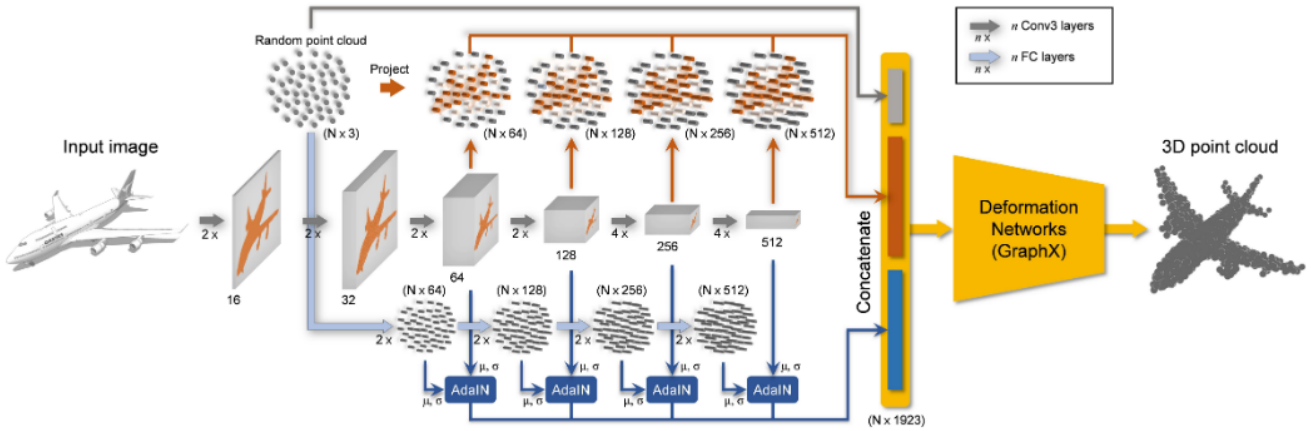


Figure 2. Overview of PCDNet. The network consists of three separate branches. Image encoding: this branch (middle) is a CNN that takes an input image and encodes it into multi-scale 2D feature maps. Point-specific shape information extraction: this branch (top), which is parameter-free, simply projects the initial point set to the 2D feature maps at every scale to form point-specific features. Global shape information extraction: the final branch (bottom) is an MLP that processes a randomly generated point cloud and 2D output features from the CNN. The features and the 2D feature maps at the same scales are fed to an AdaIN operator to produce global shape features. All the features plus the point cloud are concatenated and input to a deformation network.

The overall approach is as follows:

1. Encode the input object image using a CNN to extract multi-scale feature map
2. Distill global and point specific shape information from the features and blend it into a randomly generated point cloud
3. Use the deformation network on the mixture to obtain the 3D point cloud

Image encoding

- VGG(VGG stands for Visual Geometry Group; it is a standard deep Convolutional Neural Network (CNN) architecture with multiple layers) like architecture.
- Feed-forward network without any shortcuts from lower layers
- Consists of several spatial down-samplings and channel up-samplings at the same time
- Gives multi-scale representation of the original image

Feature blending

Point specific shape information

- Extracts a feature vector for each individual point by projecting the points onto the feature maps

Global shape information

- Processes the initial point cloud with a simple multi-layer perceptron (MLP) encoder composed of several blocks of fully connected (FC) layers to obtain features at multiple scales.

Let the set of c_i-dimensional features from the MLP and the 2D feature maps from the CNN at scale i be $\mathcal{Y}_i \subseteq \mathcal{R}^{c_i}$ and $X_i \in \mathcal{R}^{c_i \times h_i \times w_i}$ (c_i channels, height h_i, and width w_i), respectively. We define the 2D-to-3D AdaIN as

$$\text{AdaIN}(X_i, y_j) = \sigma_{X_i} \frac{y_j - \mu_{\mathcal{Y}_i}}{\sigma_{\mathcal{Y}_i}} + \mu_{X_i}, \qquad (1)$$

where $y_j \in \mathcal{Y}_i$ is the feature vector of point j in the cloud, μ_{X_i} and σ_{X_i} are the mean and standard deviation of X_i taken over all the spatial locations, and $\mu_{\mathcal{Y}_i}$ and $\sigma_{\mathcal{Y}_i}$ are the mean and standard deviation of the point cloud in the feature space.

Point cloud feature extraction

- To get a single feature vector for each point, concatenate the two features with the point coordinates together.

Point cloud deformation

- Produces a point cloud representation of the input object via an NN.
- Generating a precise and representative point cloud requires some communication between points in the set
  - The X convolution fits this purpose as it is carried out in a neighborhood of each point, but it is computationally expensive as it runs the k-nearest neighbor every iteration
  - Graph convolution considers the local interactions of the vertices, but is designed for mesh representation which requires adjacency matrix
- The paper proposes an amalgamation of the two operators called graphX-convolution (GraphX) which has similar functionality as the graph convolution, but works on unordered point sets like X-conv.

Let $\mathcal{F}_j \subseteq \mathcal{R}^{d_j}$ be the set of d_j-dimensional features fed to j^{th} layer of the deformation network. For notation simplicity, we drop the layer index j and denote the output set as $\mathcal{F}_o \subseteq \mathcal{R}^{d_o}$. Mathematically, GraphX is defined as

$$f_k^{(o)} = h\left(n_k\right) = h\left(W^T\left(\sum_{f_i \in \mathcal{F}} w_{ik} f_i + b_k\right) + b\right), \quad (2)$$

where $f_k^{(o)}$ is the k^{th} output feature vector in $\mathcal{F}_o$, $w_{ik}, b_k \in \mathcal{R}$ are trainable *mixing weight* and *mixing bias* corresponding to each pair $(f_i, f_k^{(o)})$, $W \in \mathcal{R}^{d \times d_o}$ and $b \in \mathcal{R}^{d_o}$ are the weight and bias of the FC layer, and h is an optional non-linear activation.

Results

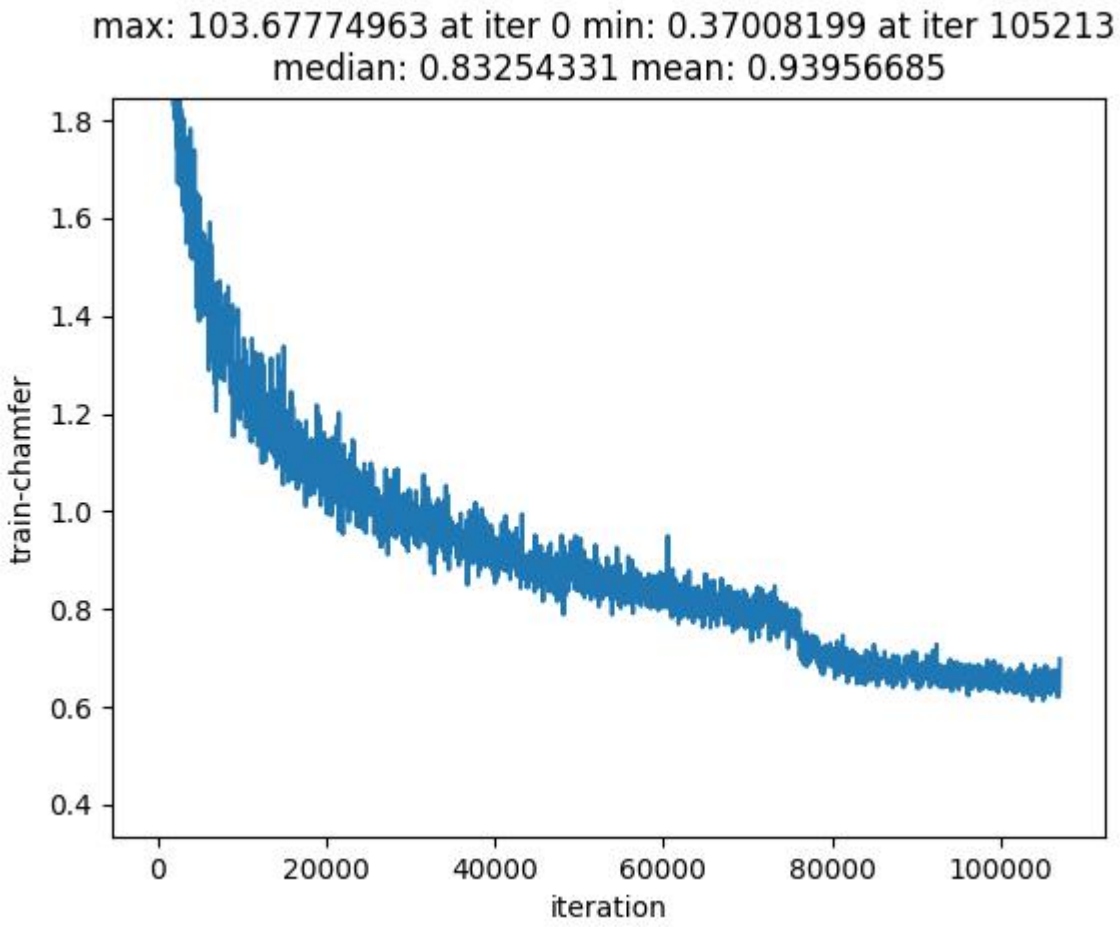
Info

Since the dataset was huge (234 GBs uncompressed), we've trained and tested our model for a subset (24 GBs) and got decent results with the default 80-20 split.

> Refer to this link to obtain the respective trained models and test results from the model.

Training Loss

For training purposes, we have opt to consider Chamfer Loss as our loss metric.



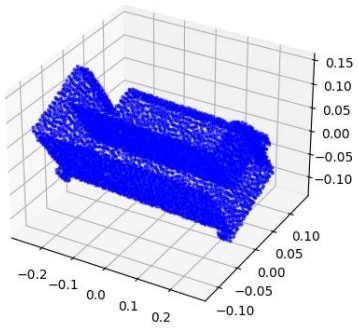
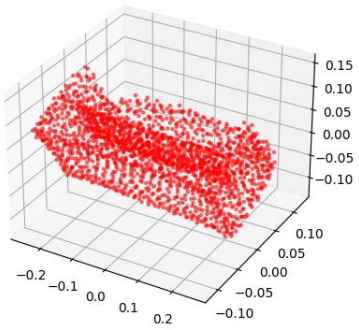
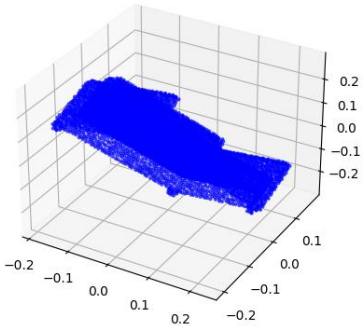
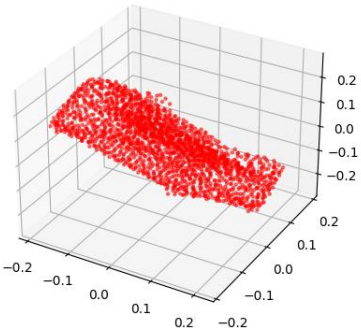
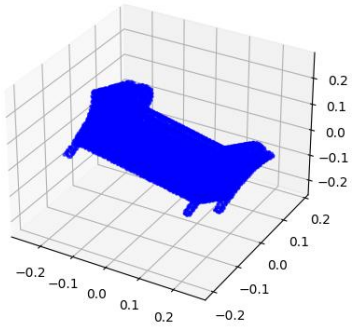
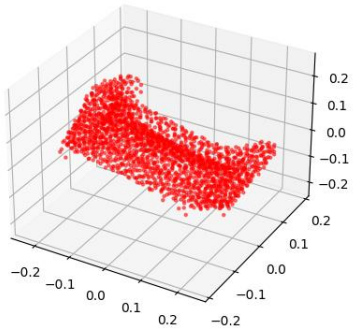
Testing Loss

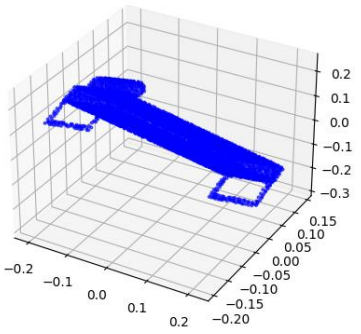
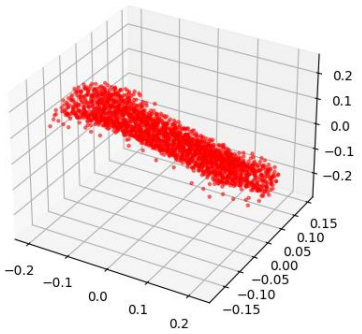
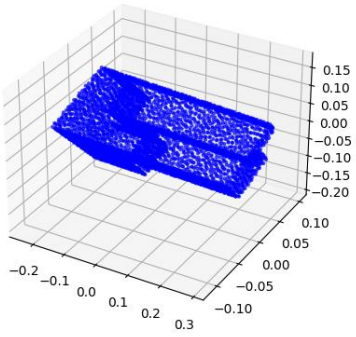
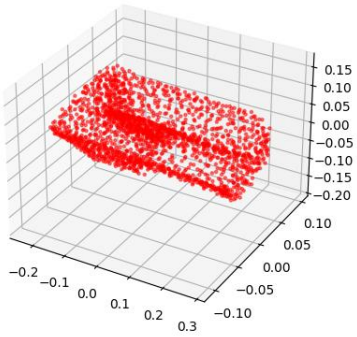
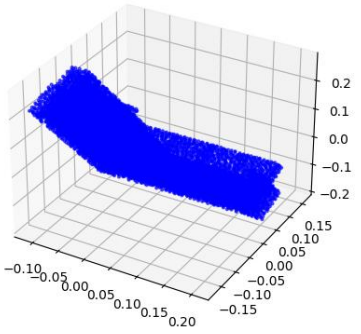
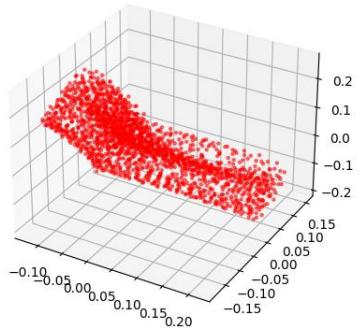
> Refer to this file for finding respective chamfer loss for the final model.

- Test loss on final model: 0.283819.

Model Output Comparision

Example	Ground truth	Model Prediction

Example	Ground truth	Model Prediction
1.		
2.		
3.		

Example	Ground truth	Model Prediction
4.		
5.		
6.		

Interactive Point Clouds

To visualize the respective models as *3D-point clouds*, kindly download the results.zip to download pre-processeed test-set point clouds and load them using visualizer.py.