Convolutional Neural Network for Point Cloud Classification

EEC 289Q - Final Project

Ahmed Mahmoud

Muhammad Awad

ABSTRACT

We introduce XYZ Convolutional Neural Network for Point Cloud Classification. An optimized model that incorporates normals and coordinates as points features. Our model achieves the same accuracy as the state-of-the-art 91.5% with much less training time.

ACM Reference Format:
Ahmed Mahmoud and Muhammad Awad. 2018. Convolutional Neural Network for Point Cloud Classification: EEC 289Q - Final Project. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

1.1 Problem Statement

In this project we are trying to classify 3D point cloud. The data set is sets of *point clouds*. Each point cloud is comprised of a set of n data points. A data point is a tuple of 3D coordinates (x, y, z) of the point in the space. Figure 1 shows few examples of the data set we have. Each point cloud is associated with a label shown under the model.

The data set (ModelNet40 [5]) consists of 9840 point clouds for training and 2468 point clouds for testing. Our task is to create a CNN model that can train such data set and achieve good accuracy within the test point cloud.

1.2 Challenges

At first glance the classification looks similar to classifying images which has been investigated thoroughly in the past couple of years. However, with a careful look at the problem and the type of input we have, we can realize the following challenges that will influence our model

Irregularity: While standard deep neural network models take as input data with regular structure (images of a fixed sizes), point clouds are irregular. This means we have no expectation on the input size (the number of points per point cloud). The reason behind this is that these point clouds either come from scan operations or hand-crafted which can not be constrained to give point cloud of fixed size.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

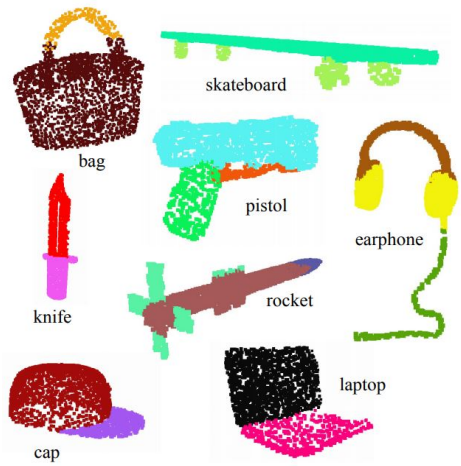

Figure 1: Example point clouds (color here are added for visualization only).

Unordered: Point positions are continuously distributed in the space and any permutation to their ordering does not change the spatial distribution. The point cloud is read from a file which could store the point in a certain order that is different than other point cloud models. Thus, associating the feature with the point order could be misleading.

Invariance: The learned features should be invariant to translation and rotation. Our model should be able to accurately classify similar point cloud even if they are rotated or shifted. Similar case exists in images where a certain feature could be located anywhere in the image. However, with point cloud in 3D space the problem is augmented as there is a third degree of freedom for translation and rotation.

1.3 Previous Work

Previous work on point cloud classification using CNN has three main directions:

View-based Methods: These methods represent a single model as a collection of 2D views to which standard CNNs used in image classification can be employed directly. View-based appraoches are good match for applications where the input comes from 3D sensor and represented as a range image in which case a single view can be used [4].

Volumetric Methods: Voxelization is another straightforward way to convert irregular data set to a regular 3D grid over which standard

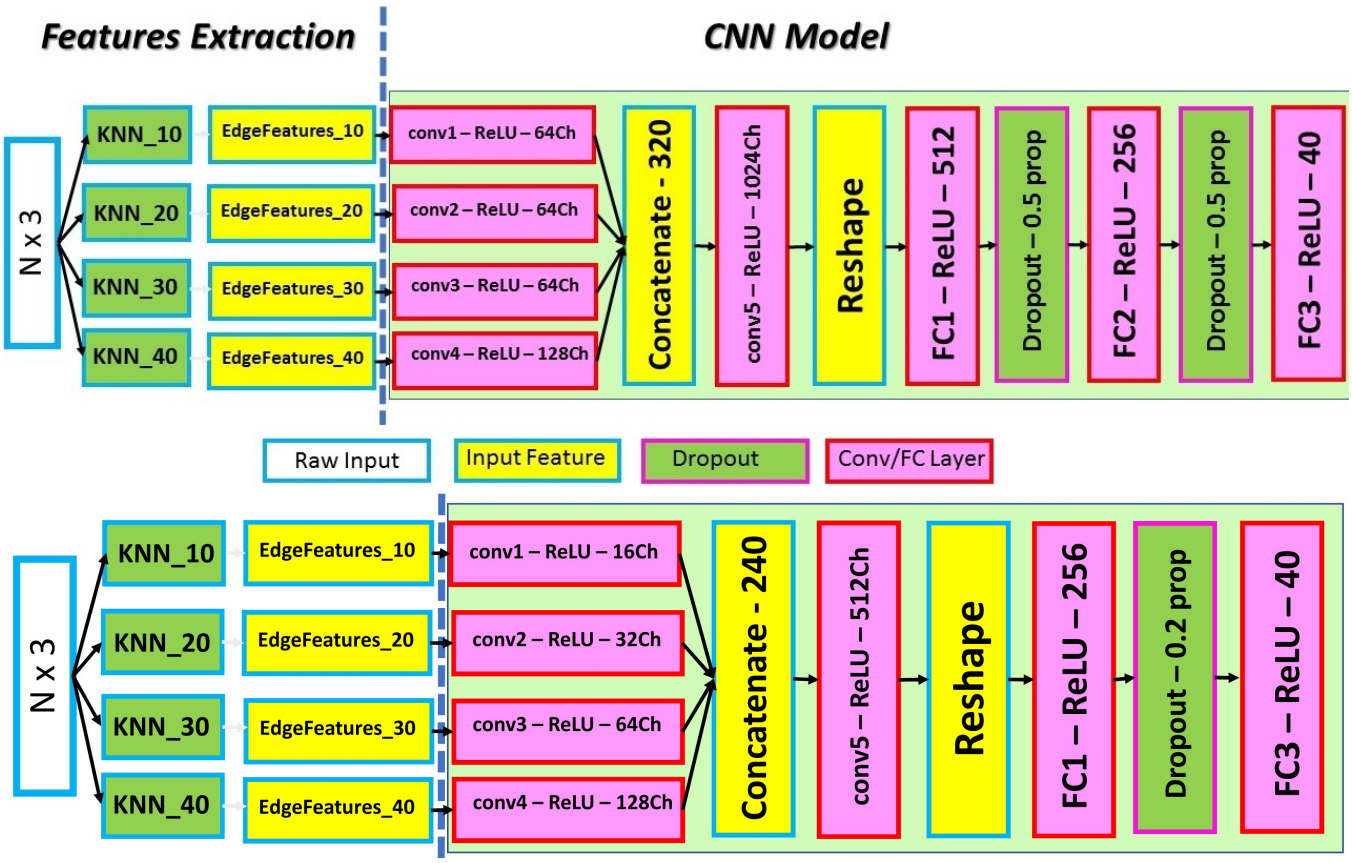

Figure 2: The main components of our model. Feature extraction is responsible of extracting global and local feature from the point cloud where the CNN model is the training model. The top figure shows the initial model, and the bottom figure shows the simplified model.

CNN can be applied [3]. However, voxelization produces a sparsely-occupied grids and induce quantization artifacts. Cleverer space partition like k-d tree can be used to mitigate these problems [2].

Direct Methods: These methods try to learn from the raw data points without voxelization or alternating the views. It should be expected that these methods to have greater accuracy since we do not manipulate the feature being learned. However, proper model design is challenging due to the challenges been mention earlier. The pioneer work on tackling point cloud learning from the raw data set is PointNet [1]. PointNet applies a symmetric function to the 3D coordinates in a manner invariant to permutation where every point is treated individually. However, PointNet employs a complex and computationally expensive spatial transformer network to learn the 3D alignment of the points. This is due to the fact that the feature being learned are global features and no local features are utilized.

2 XYZ CNN

Our model is based on dynamic graph CNN [6]. In this section we explain the architecture of our model and the rationale behind the input we choose for the model.

2.1 Model

Figure 2 shows our CNN model. The model consists of two parts; features extraction and CNN model. The features extraction tries to pick the appropriate features to learn from. The CNN model is very similar to that used in PointNet architecture in terms of the number of layers and the hyper parameters values. Later, we will explain the changes we have done to the model to speed up the training while maintain the same accuracy.

2.2 Feature Extraction

We distinguish between two type of features; global and local. Global features are the features associated with the point itself regardless to points neighbors to it. Examples include the point coordinates or point normal (vector with source at the point, perpendicular to the underlying surface and pointing outwards. See Figure 4). Local features associate the point with its neighbors. Example include the distance between the point and its K nearest neighbor points. Our hypotheses is that learning both the global and local features is essential in order to achieve better accuracy.

EdgeFeatures in Figure 2 represent the input to the CNN model such that both global and local information are well extracted from

the input point cloud. For local features, we collect the K nearest points for each point where K is a hyper parameter. We define the function that extract the features of point i as $h(x_i, x_i - x_j), \forall j \in N(i)$ where x_i is the coordinates of the point i and $N(i)$ is the local neighborhood of i (the K nearest points) [6]. The function basically concatenates the coordinates of the point along with the difference between the point and each of its K nearest neighbors. In that way, a point will have $6 \times K$ features; (x, y, z) coordinates of the point concatenated with the difference in each dimension with the coordinates of the K nearest point (Figure 3). We apply this feature extraction layer four time, each with different value of K. That way we are able to extract a hierarchy of local features that could aide the learning process to achieve faster training.

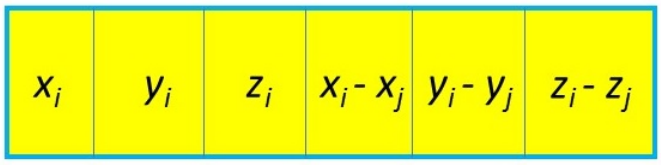

Figure 3: Features per point per point cloud. This tuple is repeated for each point K times for K nearest point where each nearest point give different (x_j, y_j, z_j) to compute the difference with.

The function as described above is able to extract the proximity in distance between a point and each of the points in its local neighborhood. We can easily recognize that this feature alone can not be very reliable. Figure 4 shows example of two pair of points belongs to two difference shapes. Even though the distance is similar, the normal associated with each point is different which can be used as indication that each pair has different underlying surface. For that, we use (in addition to the distance) the point normal during the feature extraction. We apply similar function as this applied for distance but here instead of using the difference, we used the dot product which gives the sense of how much two vectors are similar. Note that the dot product between two vectors returns a single value. Thus, a single point will have $4 \times K$ features.

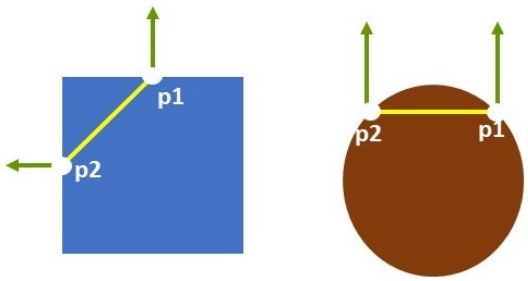

Figure 4: Distance metric alone between two points of different shapes could be misleading. Aided with point normal, extra features like curvature and shape corners could be learned.

2.3 CNN Model:

After extracting the right features, we input each feature layer into a convolutional layer. The output channel in each layer is a hyper parameter that we changed for different experiments. After that we concatenate the output of all convolutional layers into one aggregation layer. The output then goes to three fully connected layers between each there is a dropout layers. The final fully connected layer is the classification layer with 40 classes since the data set we are using (ModelNet40) contains 40 classes

2.4 Implementation:

Here we explain more details about our implementation using TensorFlow. For each input model, we pick (uniformly random) 1024 points from it to train an entire epoch. For each epoch we (uniformly random) pick different set of points. That way we can tackle the irregularity of the data set and capture different parts of the point cloud and the number of epochs increase.

Hyper parameters shown in Table 1 are fixed in all our experiments. We used batch normalization to help train faster and also help as a regularizer. After training an epoch, we run an evaluation to compute the test (out-of-sample) accuracy. We report the maximum accuracy obtained i.e., early stopping.

Table 1: Our model fixed hyper parameters.

Parameter	Value	Comments
# Points	1024	
Batch Size	32	
Max epoch	300	
Learning rate	0.001	
Momentum	0.9	for batch normalization
Optimizer	Adam	
Step Decay	200000	for learning rate
Decay Rate	0.7	for learning rate
Step Decay	200000	for momentum
Decay Rate	0.5	for momentum

3 RESULTS

We have conducted different experiments starting with the model shown in Figure 2 where we have applied several modification to it to obtain higher performance in terms of accuracy and/or training time. We started without including the point normal information. We observed a large gap between train accuracy and test accuracy indicating an over-fitting. After 200 epochs, the train accuracy was 98% while the test accuracy was 91%. Additionally, the model was too slow as it could take upto a day to complete the first 150 epochs on Nvidia Titan V GPU.

The fully connected layers were the bottleneck of our models. The huge number of weights introduced during these two fully connected layers takes up all the computation time. For that, we removed one of them (FC1). Additionally, we decreased the number of channels in each of the first four convolutional layers to be (16, 32, 64, 128). Finally, we reduced the dropout rate to be 0.2 instead of 0.5 (See Figure2). This reduced the training time significantly as

it could take 8 hours to train the first 150 epochs. The test accuracy was not affected by these changes; it became 90.07%. Surprisingly, this also helped the decreasing the gap between the train accuracy and test accuracy where the train accuracy was 87%.

Going to back to how we extracted the features, we could notice that each of the feature layers (labeled as **EdgeFeatures_XX** in Figure 2 has duplicated information. Each point concatenates its coordinates along with the difference in coordinates with one of its K neighbors. This means that the single point coordinates is being feed to the CNN model K times for each layers. For that, we concatenated the point coordinates only once when K is largest. For other layers, only the local information was feed in. This helped reducing the train time even further. It only took 4 hours to train first 150 epochs. The test accuracy slightly increased to be 90.3% and the train accuracy increased to be 94.5%

We now introduce the points normal features and feed it to the CNN model. We do this by replacing **EdgeFeatures_10** and **EdgeFeatures_30** to be **NormFeatures_20** and **NormFeatures_40**. That means we take the point and normal information from two set of neighborhood for $K = 20$ and $K = 40$. We allow concatenation of the point feature and normal feature only for layers with $K = 40$. The training time is reduced even further to be 2.5 hours for the first 150 epochs. This is due to the fact the NormFeatures tuple per point is 4 while the EdgeFeatures is 6 which reduced the number of weights. In addition, the train accuracy increased to be 90.6%. With further experimentation, we found that we can increase the test accuracy to be 91.5% if we increased the number of channels in the first three convolutional layers to be 64 while increasing the number of channels in the aggregation and fully connected layer to be 512. However, this model take longer to train; more than 5 hours for the first 100 epochs.

REFERENCES

[1] Kaichun Mo Leonidas J. Guibas Charles R. Qi, Hao Su. 2017. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

[2] Roman Klokov and Victor Lempitsky. 2017. Escape from cells: Deep kd-networks for the recognition of 3d point cloud models. In *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE, 863–872.

[3] Daniel Maturana and Sebastian Scherer. 2015. Voxnet: A 3d convolutional neural network for real-time object recognition. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE, 922–928.

[4] Lingyu Wei, Qixing Huang, Duygu Ceylan, Etienne Vouga, and Hao Li. 2016. Dense human body correspondences using convolutional networks. In *Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on*. IEEE, 1544–1553.

[5] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 2015. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1912–1920.

[6] Ziwei Liu Sanjay E. Sarma Michael M. Bronstein Yue Wang, Yongbin Sun and Justin M. Solomon. 2018. Dynamic graph CNN for learning on point clouds. *arXiv preprint arXiv:1801.07829* (2018).