Neural Networks for Classifying Noise in Point Clouds

Justin Perkins, Joe Wilder

Department of Computer Science, University of New Hampshire

CS 852: Foundation of Neural Networks

Prof. Laura Dietz

December 6th, 2024

Task

We propose a neural network that will be able to detect outliers in point clouds. Our input will be a point cloud of shape (N x 3), where N is the number of valid points plus the number of noisy points. Our model will take this point cloud and classify each (x, y, z) coordinate point as either part of the point cloud, or as a noisy data point that can be removed. This results in an output of shape N x 1. Through this classification, we will have a label for every point allowing us to remove noise from our point cloud.

To encode our point clouds to a fixed size, we will model part of our neural network on the PointNet architecture. This model makes it possible to encode point clouds to a lower, fixed dimension which will be required for the input of our model. It can also help with the training speed and robustness of our model by not considering the ordering of the points in the point cloud.

Motivation

Point clouds are used in many domains, and processing point cloud data introduces many challenges. Our model is attempting to find a more optimal way to process point cloud data. It is important to solve this task because point clouds are commonly used for downstream tasks such as 3D reconstruction and data visualization. When noise is introduced into 3D reconstruction and other visualizations, the results will be less optimal. Manually removing points by hand is also too time-consuming. Having a model to automatically perform this process would help with this. Overall, the experience for people using this model could save time and get better results for downstream tasks.

Dataset

We will use a synthetic dataset created from the ModelNet10 dataset.[1] This dataset consists of around 4000 mesh files of various objects. To save time manually annotating point clouds, we will perform sampling on the mesh files to extract a point cloud. This gives us a ground truth label. To get a corresponding input we will introduce various types of noise to destruct our ground truth point clouds. We have successfully downloaded this dataset and inspected the 10 different classes. They include objects such as chairs, sofas, and beds. To get training and test splits, we will separate the dataset into subsets. 70% will go to training, 25% to test, and 5% to validation. In addition to this, there are more ways that we could split the data. One method we could take is to fully exclude one of our classes from training. This will allow us to see if our model can perform well on objects that it has never seen before. Initially, we will start with training a model on all objects. We will also plan to train a second model without one of the classes so that we can see how the results compare, and how well the model generalizes to unseen shapes.

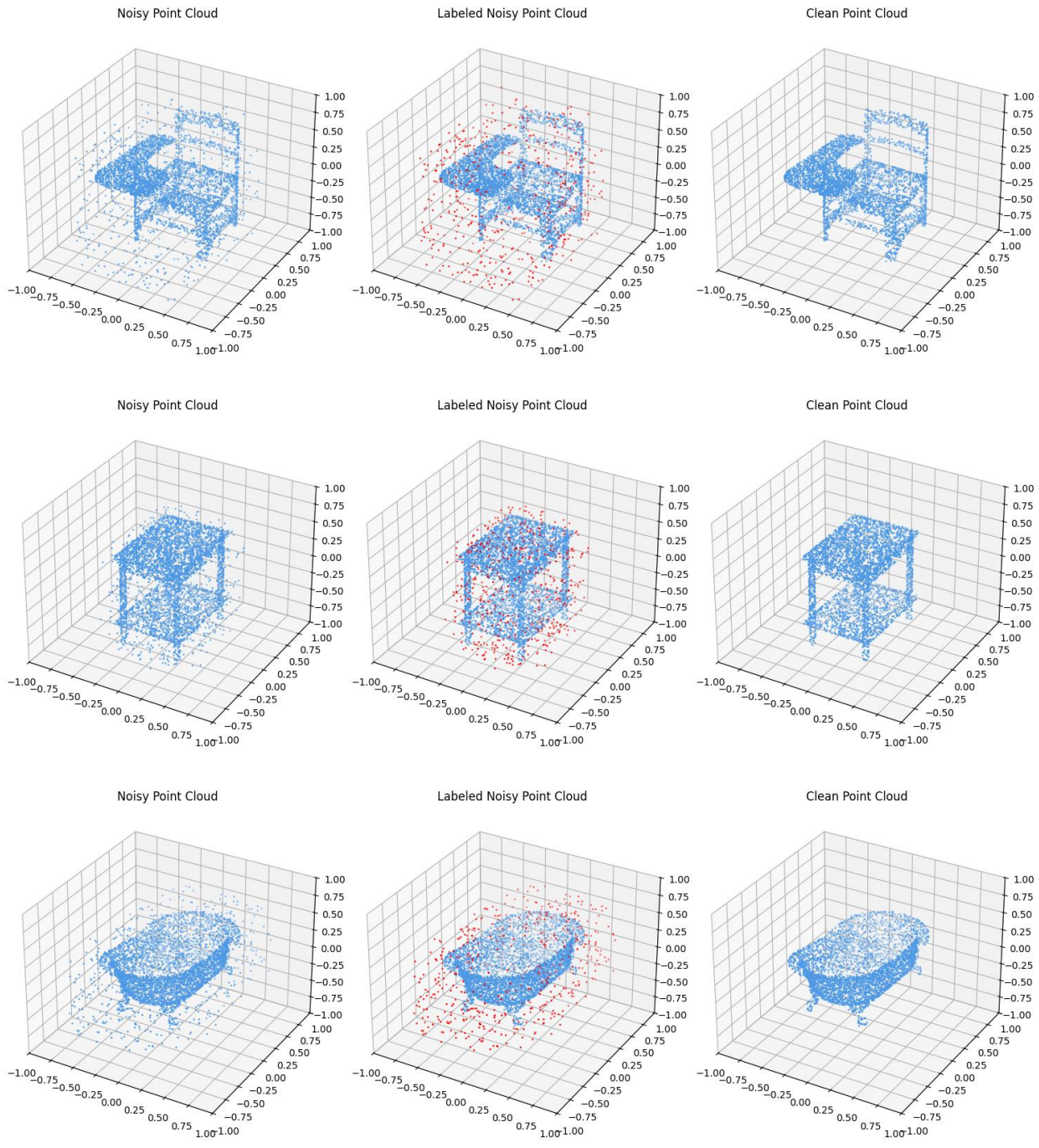
Our dataset is made up of a series of point clouds that correspond to various objects. We need to customize the data set by introducing noise across the point clouds. By using a synthetic dataset, we avoid the need to manually annotate our own data. In addition, if PointNet is not used to create a fixed feature vector, we need to sample the same number of points from each example point cloud. This step is crucial to creating a good model. When creating artificial noise, we want to try and make it as close to real world noise as possible. To do this we will need to follow a few steps. The most important step is that the noise cannot be truly random. If we added fully random noise points, there is a chance a noisy point could be placed in a good spot that should not be

[1] The dataset can be found at the following http://3dvision.princeton.edu/projects/2014/3DShapeNets/ModelNet10.zip

considered noise. This would lead us to a dataset where we have points being classified as noise while in reality, they are not. Our model would get very confused and have a hard time learning with this poor data. One way to combat this is to generate a bounding box that encompasses the entire point cloud. We then randomly place noise with the restriction that it must be outside of the bounding box. To further refine this, we could use the mesh as a bounding box. By restricting the points we place to outside of the mesh, we ensure all points that are placed are true noise points.

Another issue is we need a lot of variation in our data. When collecting point cloud data in the real world using technologies such as lidar and sonar, it is common for some areas of a point cloud to have more data then other sides. Our dataset should attempt to follow this pattern, meaning our clouds should not be uniform in their point distribution. Again, in the real world, noise can also be scattered in different groups. When generating noise, we should try to have noise all over the place and also have different clusters. This will make the noise patterns harder to detect for our model and more realistic.

Here are some example images of what our dataset could look like, although keep in mind we have not added in different densities and we haven’t added in noise clusters yet.


Noisy Point Cloud
Labeled Noisy Point Cloud
Clean Point Cloud
Noisy Point Cloud
Labeled Noisy Point Cloud
Clean Point Cloud
Noisy Point Cloud
Labeled Noisy Point Cloud
Clean Point Cloud

Standard Neural Network Models

Stacked Neighbors MLP Model – Justin

For my point cloud noise classification model task, I am using a multi-layer perceptron model because it is a very simple architecture with minimal degrees of freedom. Thus, training will take less time with such a big dataset. The model takes in the individual (x, y, z) points and outputs a prediction as to whether or not it is noise. The only modification I am making to the base model is adding the n nearest neighbors to each point. This is required to make out base model feasible as our model simply cannot make accurate predictions based off a single (x, y, z) coordinate. The degrees of freedom for this network is 3N + 3. 3 degrees for each neighbor, and 3 degrees for the current point.

```
Def Stacked MLP
        N = number of nearest neighbors

        Stack the nearest neighbor onto the point
        Pass through Linear layer with input features 3N + 3 and output features of 1
        Pass through ReLu layer
```

Pointwise Convolution Model - Joe Wilder

For the point cloud noise classification task, I am using a pointwise convolutional model. The reason for using this is because it is a more simplistic version of the PointNet architecture. By using pointwise convolution, we can extract features for each point and embed these features into higher dimensions. After extracting this information, it can then be projected into a lower dimension through more convolution layers. By having a kernel size of size 1, it is possible to keep spatial information intact while the dimensions are changed around. The model consists of an input of B x N x 3, where B is the batch size, N is the number of points in the point cloud, and 3 is the x y z values. This input is passed into 3 separate convolution layers to extract features.

First we start with the dimensions of 3 which we get from the coordinates, and each convolution expands those features first to 64, then 128, then 1024. After that, we run it back down from 1024 to 512, 256, and finally down to a single prediction again using convolution layers. The dimensionality of the layers was chosen because it mirrors the PointNet's feature extraction layers. The model is appropriate for the task because it can learn what makes a valid point cloud shape, and remove points that are not part of that shape. Pointwise convolution processes each point independently, which can make them more flexible like a MLP. One thing to consider, however, is it may struggle with the data as it focuses more on the global structure of the point cloud rather than using local neighboring features.

```
1) Take input of (B x N x 3), which represents a batch of point clouds
2) Run point clouds through a convolution layer, then into relu activation function
3) Expand dimensions using multiple convolution layers
4) Finally, reduce dimensions to a size of 1, where we have N classifications
```

The degrees of freedom for this model is 256 + 8,320 + 123,096 + 524,800 + 131,328 + 257 = 788,057 degrees of freedom. This gives the model many weights to learn which allows it to capture the complexity of the task, but not enough to overfit on the amount of data that we have.

Custom Neural Network Models

Stacked Neighbor Position Wise Feed Forward MLP Model – Justin

For the custom approach, I wanted to further explore the data we can extract from the neighbors. Additional information like distance from the current point, and estimated labels for the neighbors could be critical information to making accurate predictions. For example, if we know that the nearest neighbors are all noise, we might infer with a higher certainty that the

current point is also noise. This sub-prediction will be made by using a position-wise feed layer, and the distance will be extracted from a distance matrix in which all points are compared to each other. The degrees of freedom for this network is 5N + 3. 3 degrees for each neighbor plus a degree for the distance and a degree for the label prediction. As well as 3 degrees for the current point.

```
Def Stacked MLP with PFF
        N = number of nearest neighbors

        Stack the nearest neighbor onto the point
        Add the distances between the neighbors and the point

        Make an initial prediction on the neighbors with a Position Wise Feed Forward Layer
        Add the initial predictions on to the dataset

        Pass through Linear layer with input features 5N + 3 and output features of 1
        Pass through ReLu layer
```

PointNet Model – Joe Wilder

There are some additional issues with the standard convolutional approach. One of the main issues is with the point cloud data structure itself. A point cloud is made of many points and can be thought of as a list of x y z coordinates. Something to consider, however, is that a single point cloud can be represented in thousands of different ways. Our list of points can be shuffled randomly, and yet we will still have the exact same point cloud. We can also apply scaling, rotations, and other transformations to our point cloud but that will not change the overall shape and structure. This can make it much harder for a neural network to learn. To address this, PointNet can be implemented (https://arxiv.org/pdf/1612.00593). PointNet is an architecture that takes in a raw point cloud as input and uses fully connected layers to learn a transformation matrix. This layer is called the tnet layer. The learned transformation matrix from the tnet can be applied to the point cloud to change it into a space that leaves it invariant to rotations and transformations. This has been implemented because it allows our model to have a better

understanding of what a point cloud is, as it won't have to learn as much information about how rotations play into this task. After learning the transformation matrix and applying the transformation matrix to the point cloud, we then have two MLPs. One will take in the points directly and will learn additional features for each point. This can be referred to as the local features of the point cloud. This can be thought of as having a feature vector for each N point in our input. By learning features about each point, we are able to better understand information at a lower level. The second MLP can take in the local features and use max pooling to find the global features of the point cloud, allowing us to get an idea of noise at a higher level. Both the local and global features are then combined and passed into one final MLP. This layer learns from this information and classifies each point as noise or not noise.

```
1) Model takes input of (B x N x 3) once again
2) Input is passed into the TNET layer, which learns a transformation matrix to be applied to
   the point cloud to make the points invariant to rotations and transformations
3) Transformed point cloud passed into MLP 1, which learns local features
4) Global features are obtained by running max pooling on the local features
5) Local features and global features are concatenated together
6) Feature vector passed into final MLP to classify each point
```

The degrees of freedom for this model can be calculated as follows:

Input Tnet	Feature Tnet	Local MLP	Global MLP	Output MLP	Total
798,409	878,144	41,152	722,176	362,985	2,802,866

With a total of 2,802,866 degrees of freedom, this model is more complex than any of our previous models. With this amount of complexity, this model is more reliant on a large dataset, and it can take longer to train. However, with this complexity we can best capture the complexity of what goes into deciding if a point is noise or not.

Existing Solutions

There are other methods that attempt to solve this problem already. One is the SOR algorithm in the software CloudCompare. It works by computing the average distance of each point to it's neighbors. Points are classified as noise if the distance is further than the average distance plus a number parameter multiplied by the standard deviation. This is also called radius filtering and it can be implemented in Python. One issue with this method is it can require heavy tuning of parameters to get the output you want. If you want to denoise point clouds in bulk and they have a significant amount of variation, you may not be able due to the need to tune parameters for each point cloud.

Expectations

I expect that our model would work best with an implementation of the PointNet architecture. Without some variant of PointNet, our model could have a hard time learning due to the ordering of points. In reality, a point cloud can be made up of points in any order. Switching the order around will not change the result at all. Ideally, our architecture would use some kind of embedding to take out the variations in the ordering of points. This would lead to more stable point cloud representations, allowing our model to better handle point clouds it has not seen before.

Evalutation

For evaluation, the main metric we will be using to quantify if our model is successful or not will be accuracy. For each point that is classified, we will compare whether the point was

correctly classified or not. However, accuracy alone is not enough to decide whether a model is successful or not. To get a better understanding of how our model is performing, we utilize a visualization framework to plot our cleaned-up point clouds in an interactive environment. This will allow us to visualize the accuracy in a more meaningful way. By calculating the accuracy score for all methods, we will be able to have a numerical score showing which approach did best. The main improvement we will be looking for is an increase in accuracy. The more accurate our model performs, the better the model will be. In addition, visualizing the point clouds will give us an intuition as to how well the models are performing.

	Train Accuracy	**Test Accuracy**	**Binary Cross-Entropy**
MLP	0.8038	0.8038 ± .0000	0.4944
MLP with PFF Pre-Predictions	0.8038	0.8038 ± .0163	0.5219
Pointwise Convolution	0.8669	0.8606 ± .0015	0.3263
PointNet MLP	0.9609	0.9629 ± .0006	0.1037

As you can see in the table above, all models get an accuracy of more than 80%. This is an important number because it closely mirrors the ratio of clean data points to noisy data points. By learning to classify all points as clean points, our models can already get an accuracy of around 80%. Thus, models that produce an accuracy of around 0.80 learn nothing about the relationship between features in practice. This is especially true for the MLP model which has a standard error of 0. Meaning that all of its predictions were the same.

The worst-performing models by far were the basic MLP model, and MLP with pre predictions. These models ended up being too simplistic for the task. They both consist of one linear layer, but with more time and GPU compute time different variations could have been

explored and possibly surpassed our better models. That being said, increased layers means increased parameters, and a simple model with quick training time can quickly spiral to a large model that takes days to train. Striking the balance is critical.

The model that performed the best was the PointNet MLP model. This is because it extracts local features and global features for every single point, giving neighboring context and global information. The TNET implementation also helps to lead to more stable point cloud representations. From our results, we can conclude that while the actual model decision to classify a point can be as simple as a multi-layer perceptron, the feature extraction done on point clouds must be thought out very carefully to get good results.