Akirakan Challenge Task (Shape Recognition)

This is a report on the steps and actions I took to accomplish the Point Cloud Classification task.

Problem statement:

1. To set up a training/inference pipeline for shape recognition of 3D point clouds.

Objectives:

1. To develop a deep learning based model to classify the category(shape recognition of 3D point cloud) of the given input point cloud.

I tackled the task in 4 main steps which are **data analysis, model analysis, architecture's modification and model evaluation**.

1. Data Analysis

1.1 ModelNet40

The dataset used in this task originates from ModelNet40 dataset which contains 12,311 pre-aligned shapes from **40 categories**, which are split into 9,843 (80%) for training and 2,468 (20%) for testing.

The 40 categories includes: `airplane`, `bathtub`, `bed`, `bench`, `bookshelf`, `bottle`, `bowl`, `car`, `chair`, `cone`, `cup`, `curtain`, `desk`, `door`, `dresser`, `flower_pot`, `glass_box`, `guitar`, `keyboard`, `lamp`, `laptop`, `mantel`, `monitor`, `night_stand`, `person`, `piano`, `plant`, `radio`, `range_hood`, `sink`, `sofa`, `stairs`, `stool`, `table`, `tent`, `toilet`, `tv_stand`, `vase`, `wardrobe`, `xbox`.

ModelNet40 dataset contains **synthetic object point clouds** and is popular as it contains **various categories, clean shapes, and is well-constructed**.

Note: The dataset 2,048 points sampled ModelNet40 dataset given and used in this task only contains 9,840 shapes for training, not 9,843 in official.

1.2 Analysis of ModelNet40

To understand the data, I **visualize the point clouds** in it for all the 40 classes. As the labels given in the dataset are in integer, thus, I visualize and display all the classes by integer(for example, Class 0, Class 1, Class 2, …, Class 39) corresponds to each class (from airplane,bathtub,bed,..., xbox) in this work.

Visualized point clouds in given ModelNet40 dataset:

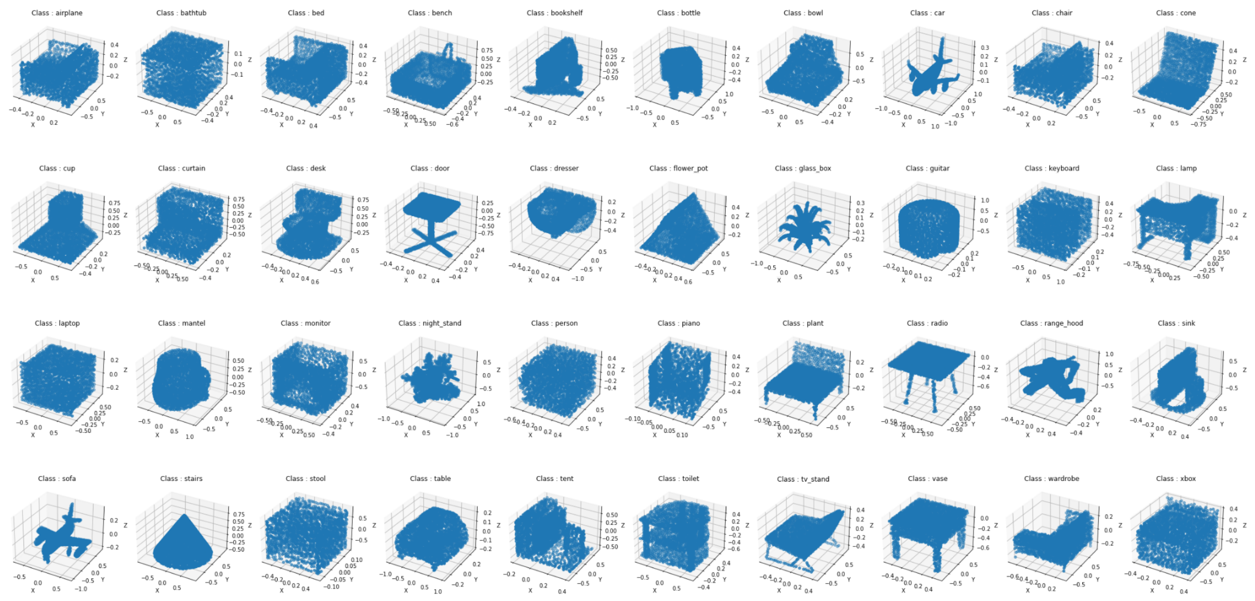


Figure 1: Visualized point clouds for ModelNet40 dataset

Beside, I also check for the **occurrence of each class** to determine whether it is a balanced dataset.

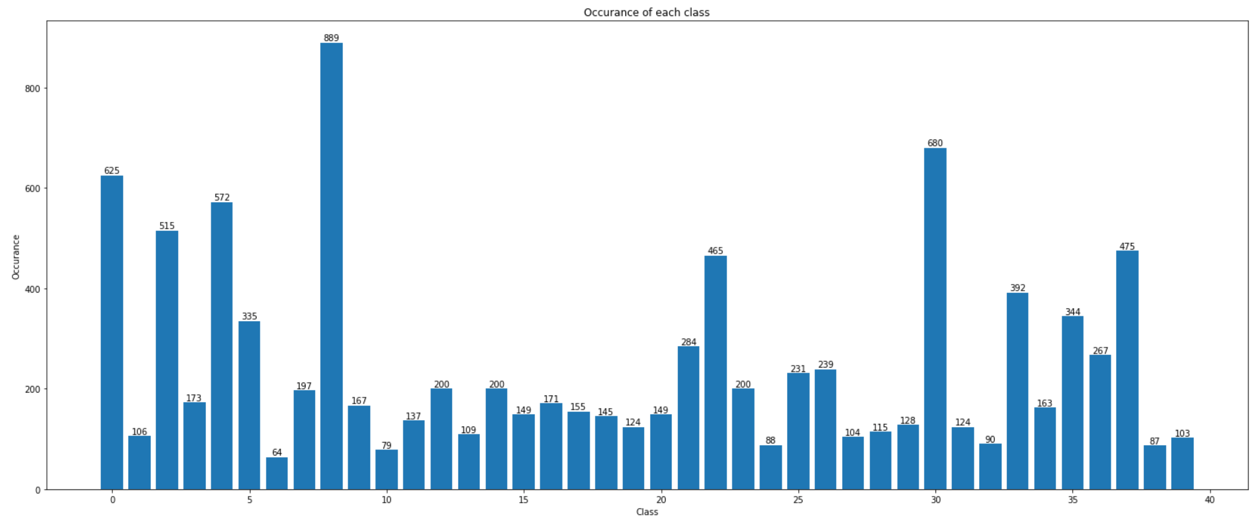


Figure 2: Occurrences of each class in ModelNet40 dataset

As shown in Figure 2, I found that the dataset is an **imbalanced dataset** where the target classes have uneven distribution of observations; with the Class label 8 has the highest number of observations(889 point clouds) while the Class label 6 has the low number of observations(64 point clouds). Thus, I carried out resampling(undersampling) and applied weighted cross entropy loss to counter this issue, which I will explain in detail in session 3.2.

1.3 3D data properties

- Sparse data(have holes, hollow space)
- Black color object has lesser points becs will be reflect away if in dark
- Might hv the issue on non-manifold geometry
- Lack of texture
- Have background noise
- Difficult to tackle/recog/detect transparent, reflective surface

1.4 Point cloud dataset properties

- Geometric data structure
- Irregular format (as input representation for deep learning network)
- Close to raw sensor data (provides more information)
- Canonical
- Unordered
    - Unlike pixel arrays in images or voxel arrays in volumetric grids, point cloud is a set of points without specific order. In other words, a network that consumes N 3D point sets needs to be invariant to N! permutations of the input set in data feeding order.
- Interaction among points
    - The points are from a space with a distance metric. It means that points are not isolated, and neighboring points form a meaningful subset. Therefore, the model needs to be able to capture local structures from nearby points, and the combinatorial interactions among local structures.
- Invariance under transformations.
    - As a geometric object, the learned representation of the point set should be invariant to certain transformations. For example, rotating and translating points all together should not modify the global point cloud category nor the segmentation of the points.

2. <u>Model Analysis</u>

2.1 Problem background

Point cloud is an unordered set of points which prevents usage of CNN object classification algorithm as CNN assumes inputs to be in an ordered data structure. Thus, the point cloud is usually transformed into an ordered representation to provide the benefit of applying convolutions. There are different ways to represent LiDAR point cloud before feeding it into the network for training if we want to use 2D Convolution operations or 2D Object Classification network such as putting it in the form of range images or bird-eye-view feature maps.

As the 3d geometric data structure point cloud is in irregular format, most researchers transform such data to regular 3D voxel grids or collections of images. This, however, renders data unnecessarily voluminous, corrupting the data and being computationally expensive.

- Point cloud is converted to other representations before it’s fed to a deep neural network:

- ○ Point cloud → Voxelization (3D CNN)
- ○ Projection/Rendering (2D CNN)
- ○ Feature extraction (Fully Connected)

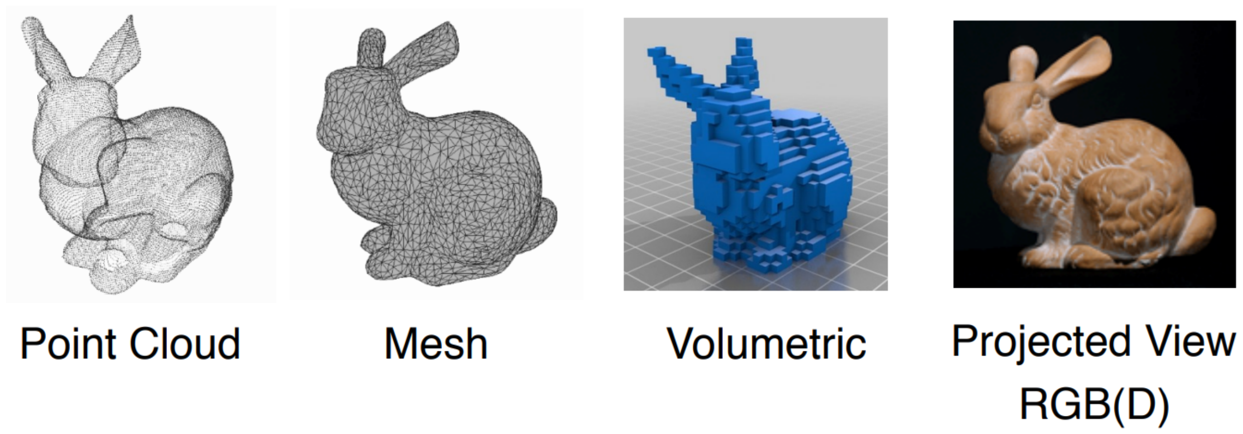


Figure 3: Different 3D data representations

Besides, most existing point cloud features are **handcrafted towards specific tasks**(for example, for object detection with basic pose estimation or the detection of partial objects),thus, not generalized enough.

2.2 Literature Review(Research on related works)

The related works on 3d point cloud classification I look into are **PointNet** and **PointNet++.**

2.2.1 PointNet

PointNet is a unified architecture that directly takes point clouds as input and outputs either class labels or per point segment/part labels of the input. The PointNet architecture as seen in Figure 4 below.

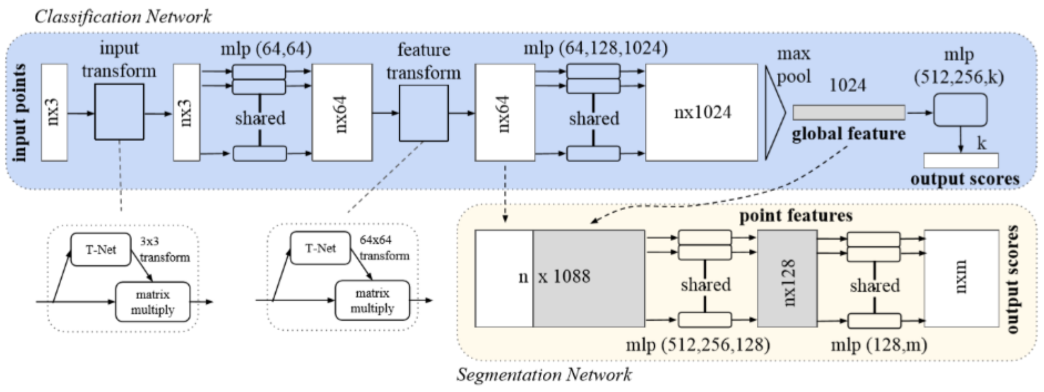


Figure 4: PointNet Architecture

The first component is a point cloud encoder that learns to encode sparse point cloud data into a dense feature vector. The PointNet encoder model is composed of three main modules :

1. Shared Multilayer perceptron (MLP)
2. Max Pooling Layer
3. Joint Alignment Network (Input transform and Feature transform)

The shared MLP layer is implemented using a series of 1D convolution, batch normalization, and ReLU operations. The 1D convolution operation is configured such that the **weights are shared across the input point cloud** to **gather information regarding the interaction between points**. It is useless to pass the weight of a single point isolated from other points as there is no useful information provided. The shared MLP layers can then learn local information of the point cloud and provide a local feature vector. Effectively the network **learns a set of optimization functions/criteria that select interesting or informative points of the point cloud and encode the reason for their selection**. The final MLP layers of the network aggregate these learnt optimal values into the global descriptor for the entire shape and feed it into the max pooling layer. We will be able to extract out the global information of the point cloud from local feature vectors and provide a global feature vector from the output of the max pooling layer.

The max pooling layer is very important because it is a **symmetric function** which ensures that the network **achieves permutation invariance** given that point cloud inputs are unordered. A symmetric function is a function that outputs the same values regardless of the order of the inputs given to it.

A **data-dependent spatial transformer network,** joint alignment network which attempts to **canonicalize (standardize) the data** before the PointNet processes them can be added to improve the results of the network. This is done by the input transform and feature transform model in the PointNet architecture. The transform model learns to predict a transformation matrix using a mini-network (T-Net) and **directly apply this transformation to align the inputs to a canonical space**. The T-Net resembles the big network and is composed of basic modules of shared MLP ,max pooling and fully connected layers as seen in Figure 5 below.

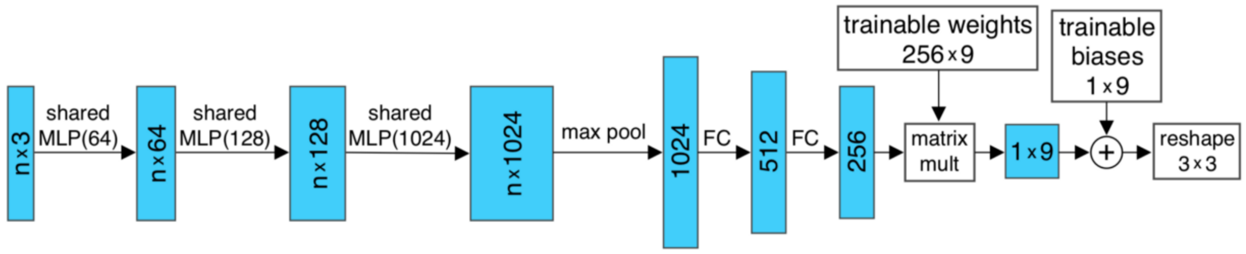


Figure 5: T-Net Architecture

The T-Net Network regresses to **find the transformation matrix that provides invariance to orientation changes** such as rotation and translation by setting up the loss function according to the property of an orthogonal matrix where $A^T = A^{-1}$, $AA^T = I$. Thus, the loss function is:

$$L = ||I - AA^T||_F^2$$

We can observed that the loss function is set up to minimize the loss so that the A matrix gets closer to that of an orthogonal matrix, where A is the 3x3 matrix of the input transform or 64x64 of the feature transform that is applied to every point cloud input through matrix multiplication.

The $||.||_F$ is the Frobenius norm. The Frobenius norm decomposes the matrix into a singular value by reshaping the matrix in a single column vector and taking the square root of the inner product of the column vector (Euclidean norm), the equation for Frobenius norm is given by:

$$||y||_F \equiv \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |y_{ij}|^2}$$

The Frobenius norm is a very important step as the loss function has to be singular value. The loss function from the T-Net is then combined with the classification/segmentation loss.

In high level explanation, is to apply transformation by T-Net to predict a transformation matrix and apply it to the input to align them into a standard representation(the author claim that this way can provide invariance to orientation) before pass to feature extraction in MLP(and the paper also proved that by doing so do improve performance of results).

The second component of PointNet after the encoder is the classifier that predicts the categorical class scores for all points or it can also be used for segmentation by concatenating the local feature and global feature vector together. Any **multi-class loss function** such as **negative log likelihood** can be used.

There are a few desirable properties of PointNet which provide it with superior results. It **processes the raw points directly, preventing information loss** from operations like voxelization or projection. It also scales linearly with the number of input points. The total **number of parameters needed** by PointNet is way **lower** than 3D Convolutions. For example, MVCNN (3D Convolution method) uses 60 Million parameters while PointNet *only need 3.5 Million parameters.*

The issue with the setup of the PointNet architecture is that it ***does not capture local structures of the points much*** *as PointNet* ***only learns the spatial encoding of each individual point and aggregates all the point features to a global feature vector****.*

2.2.2 PointNet++

Thus, the author of PointNet proposed PointNet++ to fix the issue of not capturing enough local structures. PointNet++ partitions the point clouds into different sets and applies PointNet to learn local features. The architecture of PointNet++ can be seen in Figure 6 below.

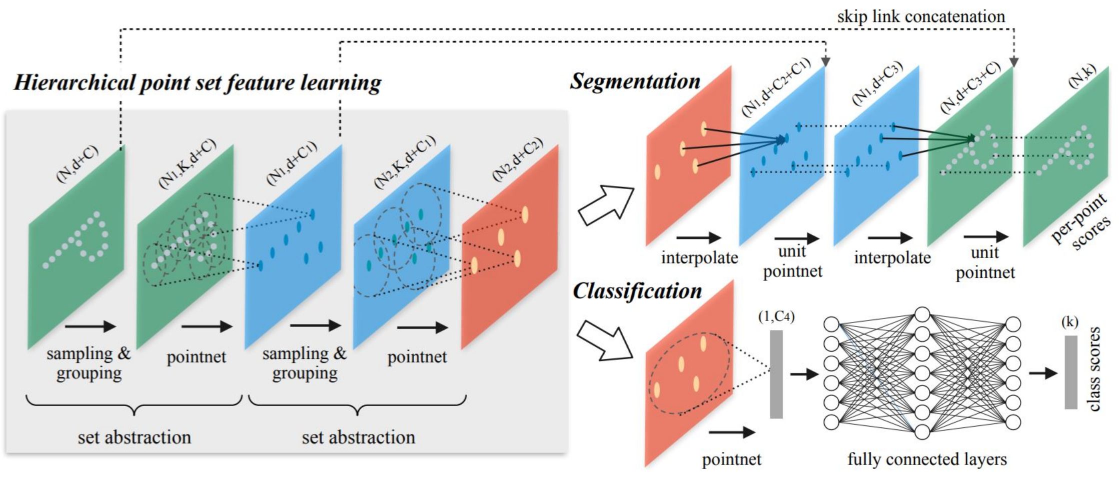


Figure 6: PointNet++ Architecture

PointNet++ architecture builds a hierarchical grouping of points and progressively extracts features off larger groupings along the hierarchy. Each level of hierarchy provides with different abstraction through Set Abstraction. Set abstraction is made of three layers:

1. Sampling layer
2. Grouping layer
3. PointNet layer

The sampling layer chooses a set of points from the input points to **define the centroids of the local region**, the grouping layer takes the centroids of the local region and **groups all of the neighboring points together to construct local region sets**. Each local region set is then **encoded into feature vectors** using PointNet. By getting feature vectors from the sets of local regions, the architecture can **get different local contexts such as edges and curves** which is one of the reasons for the performance of PointNet++ being better than PointNet. For example in ModelNet40 shape classification, PointNet++ has an accuracy of 91.9% in comparison to PointNet with 89.2%.

Therefore, after some research, I think it is **worth investigating** the “pioneer work”**, PointNet,** and, thus, **choosing it as the baseline** to accomplish this task, as numerous 3D object classification, detection and segmentation algorithms had used PointNet as the fundamental building blocks for the network.

2.3 Preprocessing (Data Augmentation)

I will use the chair point cloud object in Figure 7 as an example to show the effect of each data augmentation on the point cloud data.

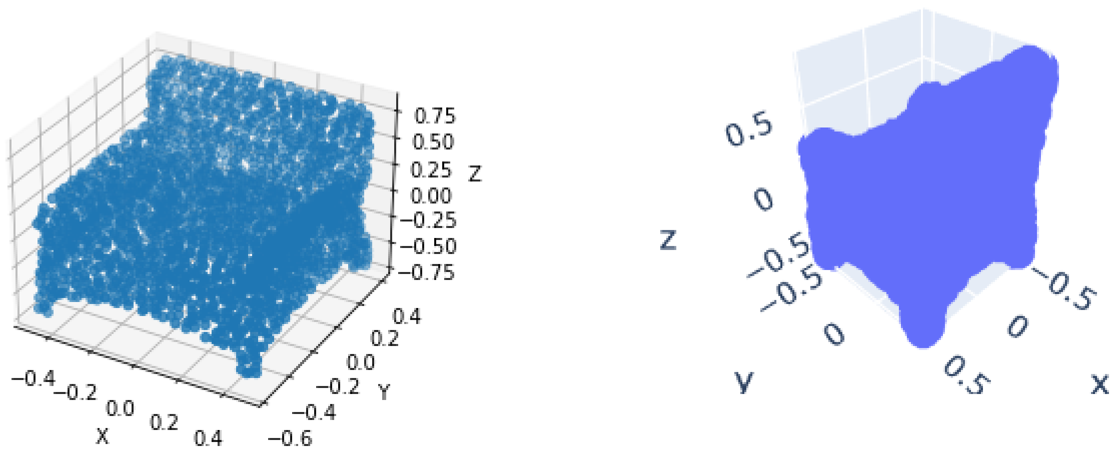


Figure 7 : A chair class point cloud object

2.3.1 Normalization

As objects can have different sizes and can be placed in different parts of the coordinate system, so we can do normalization to translate the object to the origin by subtracting mean from all its points and normalizing its points into a unit sphere (the point cloud data points are preprocessed by moving to the origin and scaling into a unit sphere). The normalized chair class point cloud object is shown in Figure 8.

2.3.2 Rotation

To augment the data during training, I randomly rotate objects around Z-axis as described in the paper. The rotated chair class point cloud object is shown in Figure 9.

2.3.3 Random Noise

To augment the data during training, I randomly add Gaussian noise as described in the paper. The chair class point cloud object added with noise is shown in Figure 10.

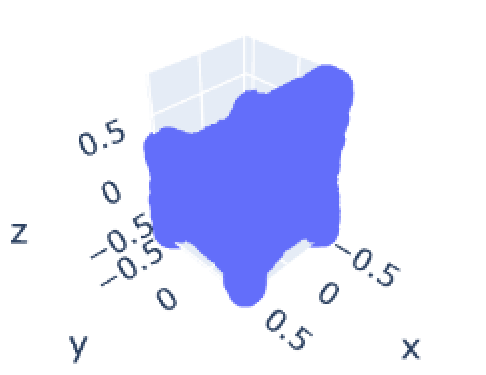


Figure 8: A normalized chair class point cloud object

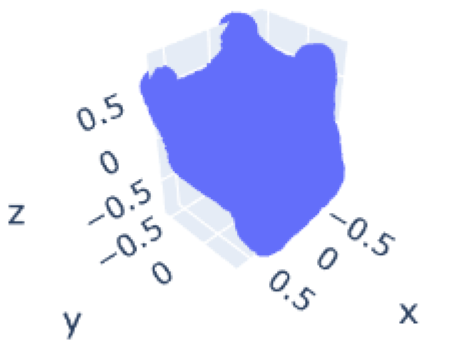


Figure 9: A rotated chair class point cloud object

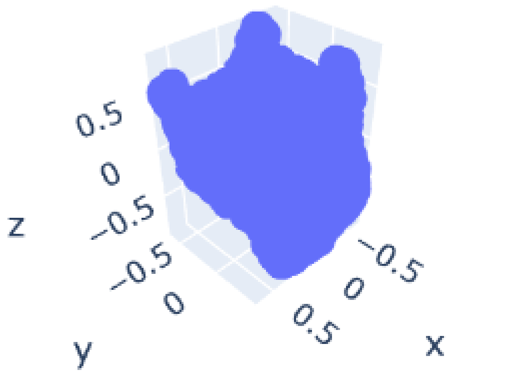


Figure 10: A chair class point cloud added with noise

2.4 PointNet Implementation

The detailed PointNet implementation can be seen in the baseline session of the all_experiments.ipynb notebook(or the_baseline.ipynb) .

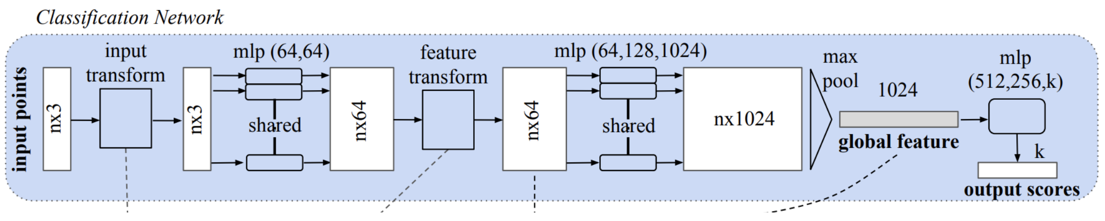


Figure 11: Classification Network of PointNet

In this implementation, only the classification network of the PointNet model as shown in Figure 11 is used.

The reasons I choose to study and implement **PointNet** is because it is :

- the **first work that consumes** and achieves effective **feature learning directly on raw point clouds data, preventing information loss** from operations like voxelization or projection, while ensuring permutation invariant.
- **end-to-end learning** for scattered, unordered point data.
- a **unified framework** architecture that is **robust** and **widely generalized for various tasks(classification, segmentation)**.
- It did various experiments on **different approaches** to **counter permutation invariance of the input point set, invariance under transformation** and showed detailed **theoretical analysis**.

#the all_experiments.ipynb notebook is located in the `'/home/meiyih/shape_recognition/ModelNet40_data/'` *directory in the server. I actually group all the experiments core cells and have all my explanations in this notebook rather than other ipynb files, but I am unable to download it down.*

3. <u>Architecture’s Modification</u>

3.1 Architecture modification

After a brief study and implementation of baseline PointNet, I personally came out with a few important highlights on,

- a good point cloud model :
    - should utilize both the **global feature** and **local feature** of point cloud data
    - should **achieve permutation invariance** in unordered point cloud inputs.
    - has a learnt representation on the point set that is **invariant to transformations**.

In PointNet, MLP(dense layer/ fully connected(FC) layer) is used a lot, because it is simply a convenient way of **making all the outputs of the previous layer contribute to the final classification**. FC layers are used to introduce scope for updating weights during back-propagation, due to its ability to introduce more connectivity possibilities, as every neuron of the FC is connected to every neuron of the further layers. For example, MLP is applied in the part after input transform and feature transform to encodes or learn local features of the point cloud; at the end of the classification network to extract out the global information of point cloud from the output of the max pooling layer for the output of classification score; and even in the T-Net of joint align network(input transform and feature transform).

*As mentioned earlier, the issue with the setup of the PointNet architecture is that it **does not capture local structures** of the points as PointNet **only learns the spatial encoding of each individual point and aggregates all the point features to a global feature vector**. PointNet lacks the ability to capture local context at different scales. Therefore, It is very **useful if the network can learn from local context like a Convolutional Neural Network (CNN) instead of just encoding each point individually**. Learning from local context at different scales helps abstract different local patterns which allows for better generalizability. For example, the first few layers of the CNN **extract simple representations such as corners,edges and spots and layers** after it builds on top of these local contexts to **form more complex representations**.*

Thus, this makes me wonder **is there any ways to increase the learning of local features without messing up the setup pipeline of PointNet architecture** and at the same time achieve the 3 characteristics that I think a good point cloud model should possess.

Research Question: Does **CNN perform better** than fully connected layers in local features extraction and invariance geometric transformation of a classification task?

Objective : To investigate the **importance of fully connected layers,mlp vs the effect on usage of CNN** which preserve spatial info on the effect for local features extraction and invariance geometric transformation in this classification task.

To answer this research question, I came to think of experiments to **replace MLP with CNN** to see if the characteristics of **CNN which learns spatial relationships between neighboring points** will help in **encoding more local features**.

This is because in classification tasks, **FC is simply a convenient way of making all the outputs of the previous layer contribute to the final classification**. **Convolutional layers on the other hand, have a sparser connectivity**, so we can't use them. However, FC layers aren't the only way to classify the features extracted by the CNN. There are also many other classifiers being used besides FC as well. That being said there are many tasks, besides classification, where CNNs are used without FC layers.

We can simulate a FC layer with convolutions. A convolutional neural network (CNN) that does not have fully connected layers is called a **fully convolutional network (FCN). An example of an FCN is the U-Net, which does not use any fully connected layers, but only convolution, downsampling (i.e. pooling), upsampling (deconvolution), and copy and crop operations.**

Moreover, we can use CNNs only for the purpose of feature extraction, and then feed these extracted features in another classifier (e.g. an SVM). In fact, transfer learning is based on the idea that CNNs extract *reusable* features.

The reason people use the FC after the convolutional layer is that CNN **preserves spatial information**. Especially with softmax, for classification tasks. Thus, my idea is to show that if I don't use MLP(means I "skipped" FC layer which combine the info from all the kernels in all positions, by converting FC to Conv net), then I will probably evaluate first class by position of the first kernel, second class by second position,and not by the whole point cloud input with all kernels. Later on, I can observe the performance to see whether it is valuable to replace MLP with CNN or not.

But the MLP layers at the part after input transform and feature transform and at the end of the classification network should not be try or modified, because these MLP layers responsible to encodes single points info which laters on aggregates for global features vector and for classification of class(an important key to tackle the unordered data point issues and achieve permutation invariant).

Thus, the only possibilities I can try out is to **modify the architecture of T-Net**(which has 1dconv in "downsample part " and fully connected layer in "upsample part", to combine all the info from all kernels) **into a Fully Convolutional Network**. Among all the CNNs, I choose to implement U-Net to replace the T-Net because U-net has skip connections which explicitly copy features from earlier downsampling layers into later upsampling layers to **help recover the full spatial information at the network output**.

This is because some information captured in the initial layers may be lost due to the depth of layers and is required for reconstruction during the up-sampling. Skip connections can bring clarity to the features recognized in the final layer. For example, small object detection is difficult in deep networks so using **skip connection would bring clarity in edges, texture, shapes etc of the features**. The advantage of skip connection is that they pass information across layers so using it to classify minute details becomes easier, which might be useful in helping for the extraction of local features in our experiment here.

Although the T-Net here in joint alignment network responsible to predict a transformation matrix and apply it to the input to align them into a standard representation to counters the transformation invariance issue, but experiments were carried out with the **hypothesis** that **CNN can extract more features, where the stronger interaction among point due to the learning approach of CNN which focus on spatial relationship**, **might form a stronger transformation matrix combination that align all input set to a canonical space and influence the feature extraction in MLP after input and features transform can encodes more local features.**

Possible outcomes:

- model might perform **worse**(if we think of CNN do build spatial relationships as compare to MLP which has symmetric fully connected layers and don't care on switching or permutations)
- model might perform **better**(if we come to think that deeper layers of CNN do learns more by extracting more feature)

Here, each approach(CNN vs MLP) has its own advantage; and the justification for each is strong and makes sense too.

# I determine the increase in local features encoding by the classification accuracy during inference.

The **modification combinations/experiment** done are :

- Modified both TNet in input transform and feature transform into U-Net.
- Modified only the TNet in input transform into U-Net.
- Modified only the TNet in feature transform into U-Net.

The idea of having experiment 2 and 3 is to observe whether only applying a CNN on the initial input point(for experiment 2) and an implementation of CNN after a MLP feature extraction,MLP before feature transform (for experiment3) will affect the final classification score.

The detailed architecture's modification implementation can be seen in the modification session of the all_experiments.ipynb jupyter notebook(or inputTrans_UNet.ipynb, featureTrans.ipynb and inputTrans_UNet_featureTrans_UNet.ipynb).

3.2 Dataset imbalance issue

The main problem with imbalanced dataset prediction is **how accurately are we actually predicting both majority and minority class**.Sometimes when the records of a certain class are much more than the other class, our classifier may get **biased** towards the prediction. Thus, I applied weighted cross entropy loss and resampling technique to solve this issue.

The detailed weighted cross entropy loss and resampling data implementation can be seen in the modification session of the all_experiments.ipynb jupyter notebook(or weighted_CE_loss2.ipynb).

3.2.1 Weighted Cross Entropy Loss

I create the weight tensor for weighted nn.CrossEntropyLoss using the formula:

$$weight\_for\_a\_class = \frac{1}{samples\_for\_a\_class} x \frac{total\_number\_of\_samples}{number\_of\_classes}$$

3.2.2 Resampling data(Undersampling)

Resampling (Oversampling and Undersampling) technique is used to upsample or downsample the minority or majority class.

When we are using an imbalanced dataset, we can :

- oversample the minority class using replacement(oversampling)
- randomly delete rows from the majority class to match them with the minority class (undersampling)

After sampling the data we can get a balanced dataset for both majority and minority classes. So, when all classes have a similar number of records present in the dataset, we can assume that the **classifier will give equal importance to all classes**.

In this work, I **apply only undersampling** which can help improve run time as compare to oversampling which increases the likelihood of overfitting since it replicates the minority class events. The implementation steps of resampling data can be seen in the modification session of the all_experiments.ipynb jupyter notebook(or undersampling.ipynb).

3.3 Data Augmentation(Shuffle)

Besides, I also try to shuffle the point cloud data to **help the training converge fast** and prevent the model from learning the order of the training. An example of a shuffled chair class point cloud object is shown in Figure 12.

The implementation steps of resampling data can be seen in the modification session of the all_experiments.ipynb jupyter notebook(or baseline_with_shuffle.ipynb)

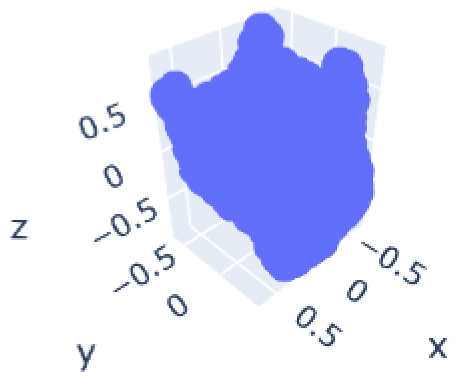


Figure 12 : A shuffled chair class point cloud object

4. Model Evaluation

4.1 Evaluation Metric

The evaluation metric I used in this task is **accuracy** and **macro F1-score**.

The accuracy of a classifier is the total number of correct predictions by the classifier divided by the total number of predictions. This may be **good enough for a well-balanced class but not ideal for the imbalanced class problem**. The other metrics such as precision is the measure of how accurate the classifier's prediction of a specific class and recall is the measure of the classifier's ability to identify a class. For an imbalanced class dataset f1-score is a more appropriate metric.It is the harmonic mean of precision and recall .

So, if the classifier predicts the minority class but the prediction is erroneous and false-positive increases, the precision metric will be low and so as f1-score. Also, if the classifier identifies the minority class poorly, i.e. more of this class wrongfully predicted as the majority class then false negatives will increase, so recall and f1-score will low. f1-score only increases if both the number and quality of prediction improves. f1-score keeps the balance between precision and recall and improves the score only if the classifier identifies more of a certain class correctly.

All the experiments I did were carried out using the original dataset distribution (imbalanced dataset without resampling), except for the experiment name baseline(with undersampling data). Thus, I include both the overall accuracy results and f1-score in Table1, where **overall accuracy** is performance measurement for **experiment** name, baseline(with undersampling data) in Table 1, **with balanced datase**t, while **f1-score** is performance measurement for the other **experiments done with imbalance dataset**.

4.2 Results

	input	accuracy	accuracy overall	F1-Score
SPH*	mesh	68.2	-	-
3DShapeNets*	volume	77.3	84.7	-
VoxNet*	volume	83.0	85.9	-
Subvolume*	volume	86.0	89.2	-
LFD*	image	75.5	-	-
MVCNN*	image	**90.1**	-	-
PointNet*	point	86.2	**89.2**	-
My baseline	point	-	79.0	**0.72**
baseline(with shuffle augmentation)	point	-	81.0	**0.72**
baseline(with undersampling data)	point	-	69.0	0.60
baseline(with weighted CE loss)	point	-	75.0	0.68
baseline(U-Net input transform and U-Net feature transform)	point	-	65.0	0.51
baseline(U-Net input transform and TNet feature transform)	point	-	57.0	0.45
baseline(TNet input transform and U-Net feature transform)	point	-	65.0	0.52

Table 1 : Classification results on ModelNet40.The results for (*) methods are obtained from the PointNet paper.

As shown in Table 1, the results of overall accuracy of baseline(with undersampling data) is lower than baseline might be because the data after resampling,which is 2560, is too few. Whereas for the experiments on imbalance dataset, the experiment of baseline(with shuffle augmentation) can achieve same f1-score as the baseline and slightly better result of overall accuracy, 81% as compared to 79% of the baseline.The experiment on baseline(with weighted CE loss also does not met the performance as expected, where it has an overall accuracy of 4% lower than the baseline. Finally, all the 3 UNet modification experiments also have their f1-score of lower than the baseline, which shows that the modification of FC layers with CNN might not be a good choice for classification task on 3d point cloud data.

Here I show a screenshot of the confusion matrix, classification report, training and inference loss, and training and inference accuracy of the baseline model. The evaluation results of the rest of the experiments can be seen in the jupyter notebook.

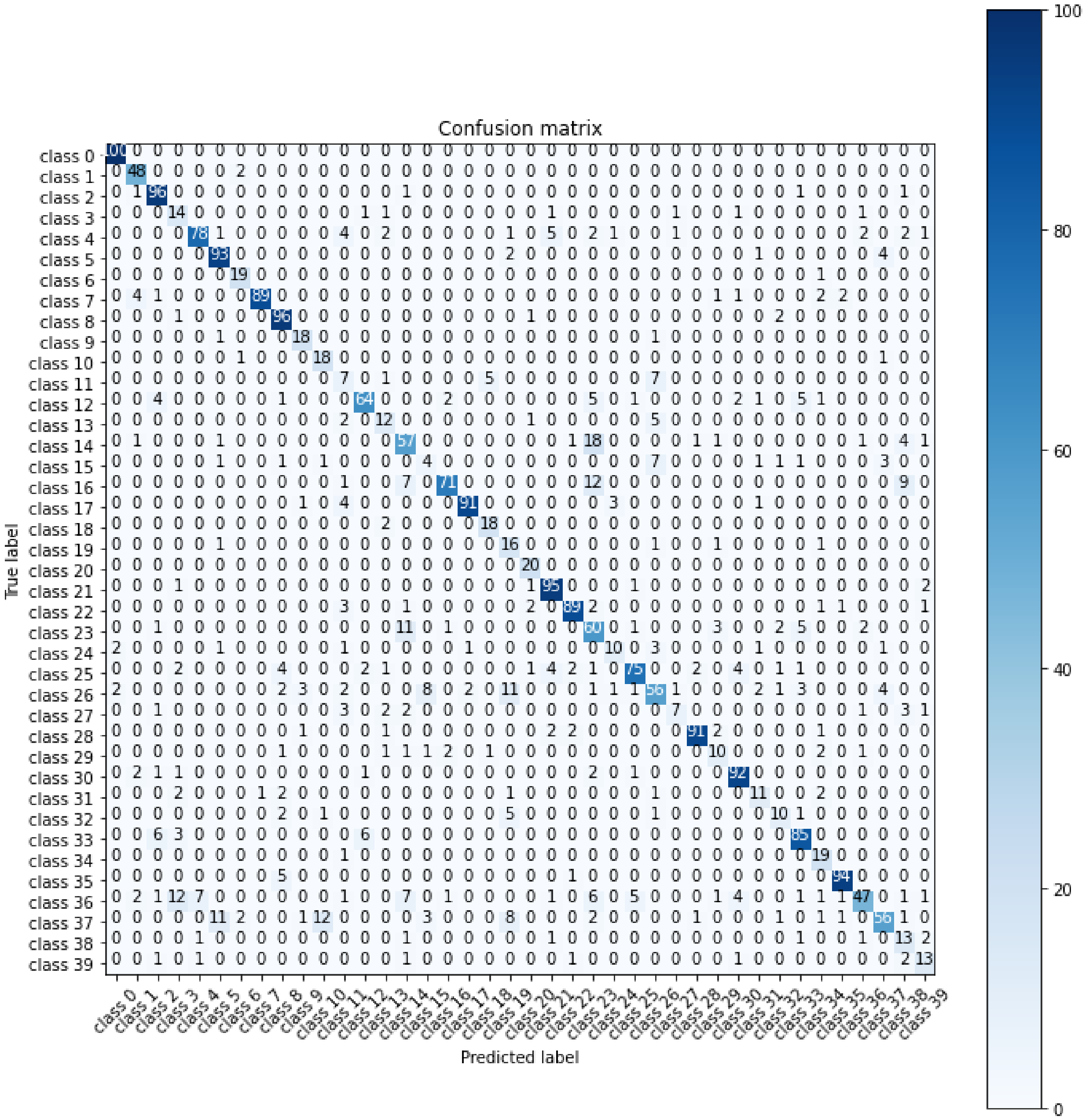
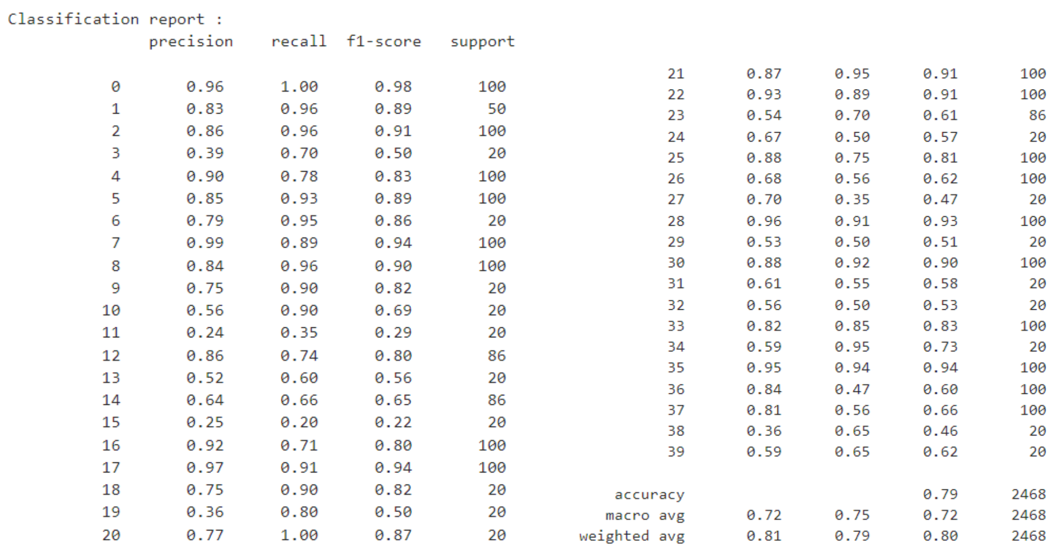


Figure 13 : Confusion matrix of baseline model



```
Classification report :
                 precision    recall  f1-score   support

           0       0.96      1.00      0.98       100
           1       0.83      0.96      0.89        50
           2       0.86      0.96      0.91       100
           3       0.39      0.70      0.50        20
           4       0.90      0.78      0.83       100
           5       0.85      0.93      0.89       100
           6       0.79      0.95      0.86        20
           7       0.99      0.89      0.94       100
           8       0.84      0.96      0.90       100
           9       0.75      0.90      0.82        20
          10       0.56      0.90      0.69        20
          11       0.24      0.35      0.29        20
          12       0.86      0.74      0.80        86
          13       0.52      0.60      0.56        20
          14       0.64      0.66      0.65        86
          15       0.25      0.20      0.22        20
          16       0.92      0.71      0.80       100
          17       0.97      0.91      0.94       100
          18       0.75      0.90      0.82        20
          19       0.36      0.80      0.50        20
          20       0.77      1.00      0.87        20

          21       0.87      0.95      0.91       100
          22       0.93      0.89      0.91       100
          23       0.54      0.70      0.61        86
          24       0.67      0.50      0.57        20
          25       0.88      0.75      0.81       100
          26       0.68      0.56      0.62       100
          27       0.70      0.35      0.47        20
          28       0.96      0.91      0.93       100
          29       0.53      0.50      0.51        20
          30       0.88      0.92      0.90       100
          31       0.61      0.55      0.58        20
          32       0.56      0.50      0.53        20
          33       0.82      0.85      0.83       100
          34       0.59      0.95      0.73        20
          35       0.95      0.94      0.94       100
          36       0.84      0.47      0.60       100
          37       0.81      0.56      0.66       100
          38       0.36      0.65      0.46        20
          39       0.59      0.65      0.62        20

    accuracy                           0.79      2468
   macro avg       0.72      0.75      0.72      2468
weighted avg       0.81      0.79      0.80      2468
```

Figure 14 : Classification report of baseline model

As we can see from Figure 14, the support column (which tells how many of each class there were) the data distribution on all the 40 classes are imbalanced. Thus, we should look at the F1 score on the performance of the model. In the classification report, the accuracy was used for a general report of model performance with balanced datasets, and the exact number that belongs to accuracy is accuracy in column of f1-score. While for our case which is an imbalanced data,only macro F1-score did a good job of describing overall model performance (caring equally about all 40 classes).

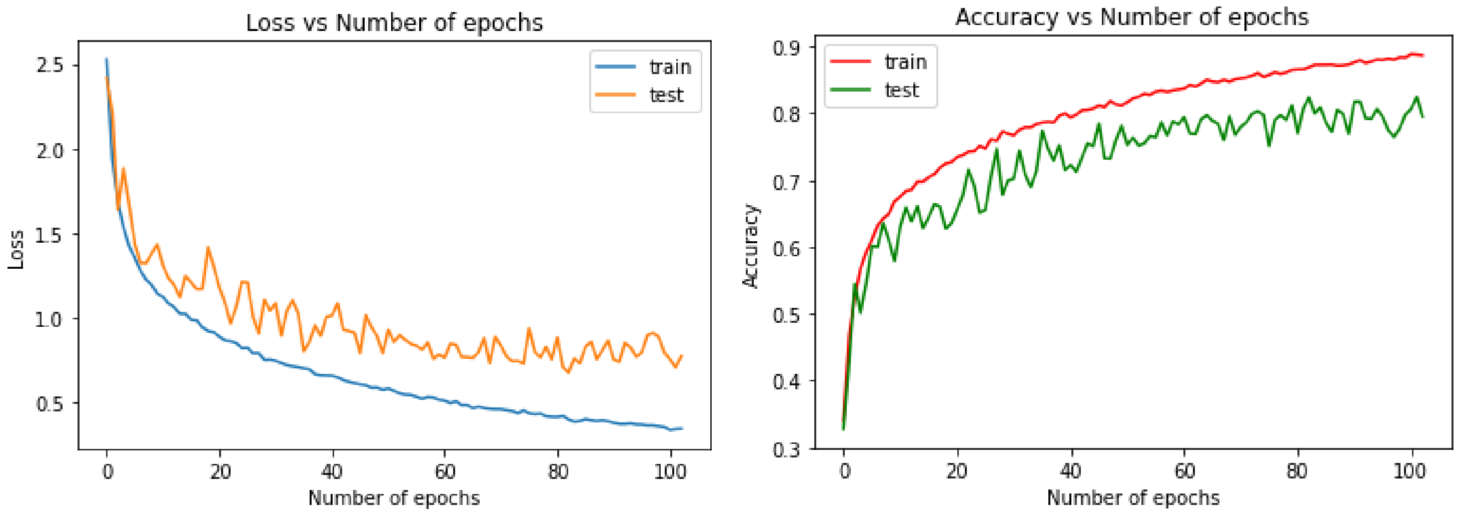


Figure 15 : Loss and accuracy graph of baseline model

4.3 Visualization results of baseline model

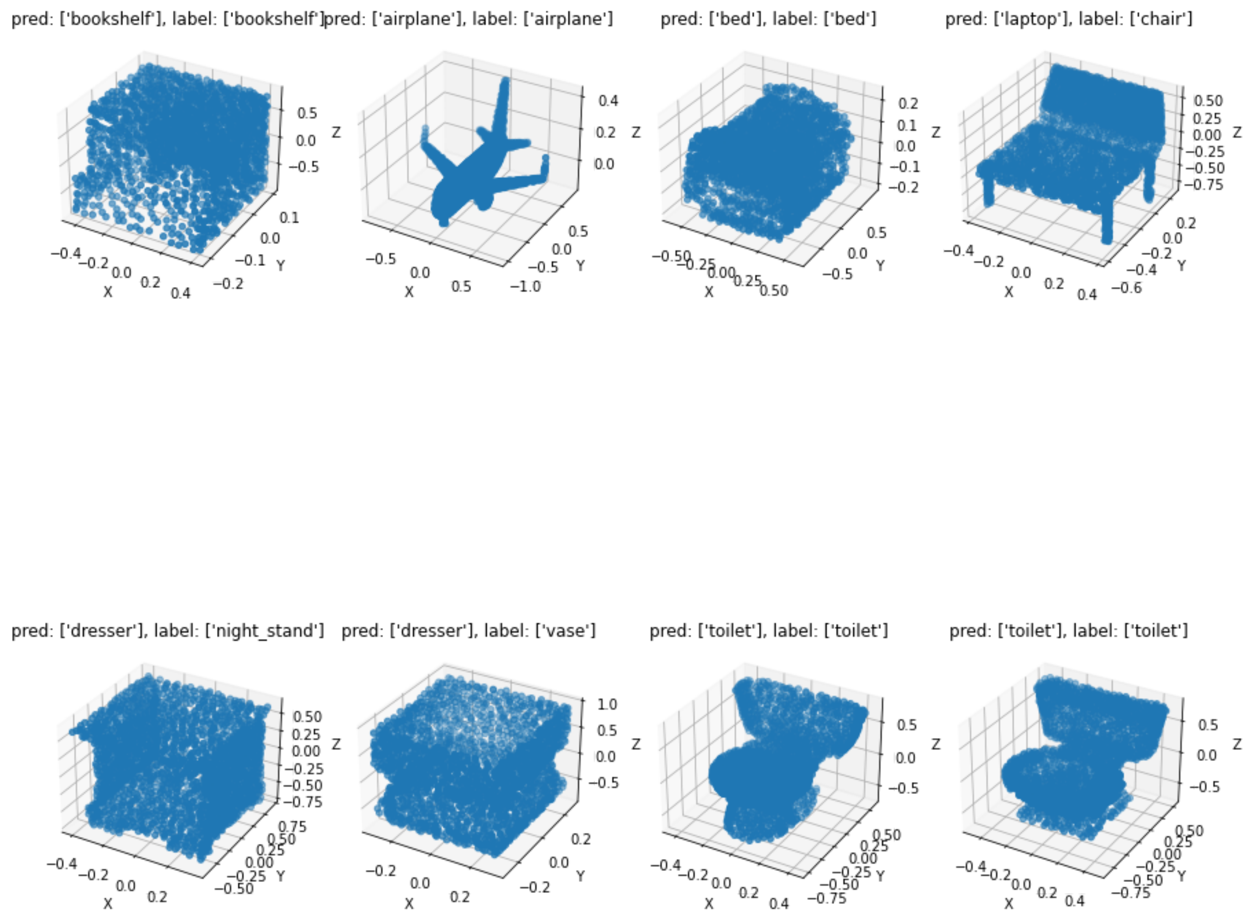


Figure 16 : Visualization of baseline model prediction

I visualized the prediction of the baseline model and found out that the model sometimes confuses dressers with nightstands, toilets with chairs and desks with tables which is rather understandable.

Future work

- normal.npy, additional data that is useful to help in learning the features better can be used. For example,PointAugment: an Auto-Augmentation Framework for Point Cloud Classification use it in their work ,https://github.com/liruihui/PointAugment/blob/5c398cfd343d683e08bb91866940e554255b4fe3/Common/ModelNetDataLoader.py#L21
- Can explore/investigate different approaches (3D point cloud operators and CNN) to solve the problem. point cloud operator https://arxiv.org/abs/2007.01294 .The aggregation methods are one of those 3D point cloud operators tailored for 3D point cloud neural networks and can be further research in future.
- #Chipping instead of resampling
    - Chipping point clouds into smaller point clouds (just like Chipping raw images into smaller tiles in 2d image processing) as an alternative way. This enables the model to learn the details, smaller or lower features of the point cloud object. However, the drawbacks might be rescaled or interpolation of points' coordinate might be needed to fill the number of points for a point cloud object (for tally dimension of matrix during training). #This is just a sudden wild thought I borrow from 2d image processing, but might not be applicable for 3d point cloud and more study need to be done to validate whether it works.
- The visualization of feature points/maps can be done to check if the modifications works

Reference

Qi, C. R., Su, H., Mo, K., & Guibas, L. J. (2017, April 10). PointNet: Deep Learning on point sets for 3D classification and segmentation. arXiv.org. Retrieved March 14, 2022, from https://arxiv.org/abs/1612.00593

Qi, C. R. (2017, June 7). PointNet++: Deep Hierarchical Feature Learning on Point Sets in a. . . arXiv.Org. https://arxiv.org/abs/1706.02413