Deep Learning – PointNet on 3D Point Cloud.

Implementation by:

Naveen Raja Elangovan

Problem Statement:

Download the ModelNet10 dataset from the below link:
http://3dvision.princeton.edu/projects/2014/3DShapeNets/ModelNet10.zip
This dataset has objects from 10 categories where each object is represented by a 3D Point Cloud. Implement a neural network architecture to solve the problem of object classification for these categories.

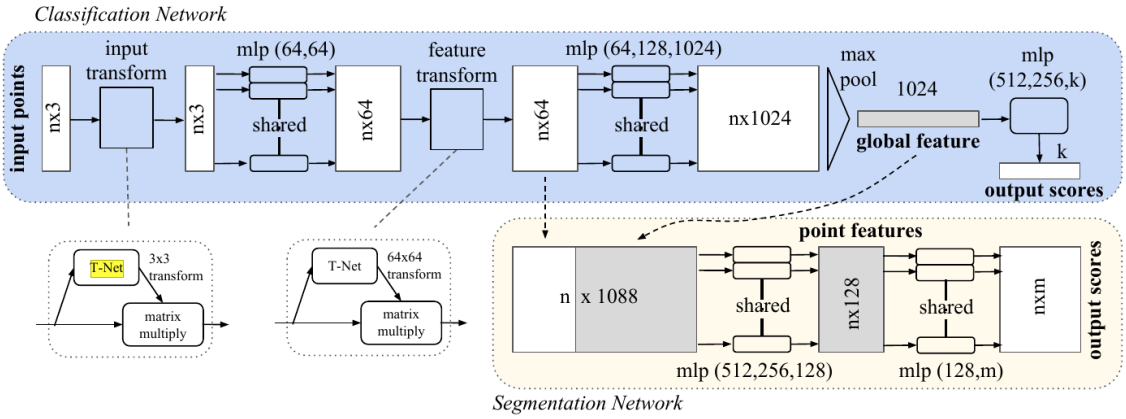
Point cloud and PointNet

A point cloud is a collection of points in a 3D space, where each point represents a specific coordinate (x, y, z) in the environment. Point clouds are commonly generated using 3D scanning techniques such as LiDAR (Light Detection and Ranging) or photogrammetry.

PointNet is a deep learning architecture specifically designed for processing point clouds. It takes as input a set of unordered points and directly operates on them, without requiring any intermediate representation such as voxels or meshes. This makes PointNet particularly suitable for handling irregular and unstructured data.

1. **Input Representation**:
    - The input to PointNet is a set of points representing a 3D object or scene. Each point is represented by its (x, y, z) coordinates, and optionally, additional features such as color or intensity values.
2. **PointNet Architecture**:
    - PointNet architecture consists of several layers of neural network operations, including fully connected layers, convolutional layers, and max pooling layers.
    - The architecture is designed to process each point independently, followed by aggregating information across all points to generate a global representation of the entire point cloud.
3. **Feature Extraction**:
    - PointNet learns hierarchical features from the input point cloud by processing each point through multiple layers of neural network operations.
    - These operations help extract local geometric features from individual points as well as capture global context and shape information from the entire point cloud.
4. **Task-specific Outputs**:
    - Depending on the application, PointNet can be used for various tasks such as classification, segmentation, object detection, or reconstruction.

- For classification tasks, PointNet typically produces a probability distribution over different classes or categories.
- For segmentation tasks, PointNet assigns a label to each point indicating the part or object it belongs to.



The above PointNet architecture has been followed by our code.

Key Limitations:

- It's important to note that point clouds are not organized in any particular order. (our algorithm should handle any possible permutation of the input set without getting confused)
- Our network should handle rigid transformations. (it should be able to identify an object even if it has been rotated, scaled, or transformed in some way). This is crucial for accurately recognizing objects in various orientations.
- our network should capture interactions among points. (it should be able to understand how different points within the point cloud relate to each other, and use this information to make more informed decisions).

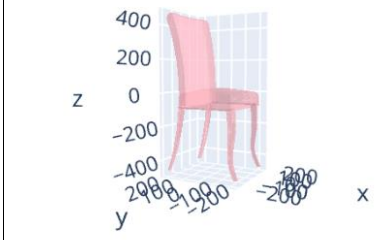
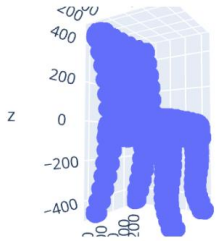
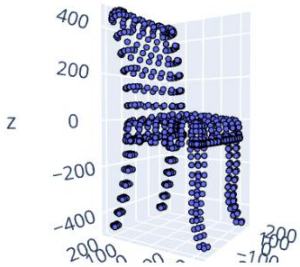
Data Augmentation techniques:

Objects come in a variety of sizes and can be positioned in various locations within our coordinate system. In order to improve our training, we also incorporate data augmentation techniques such as randomly rotating the object along the Z-axis and introducing Gaussian noise.

Train and Test data split:

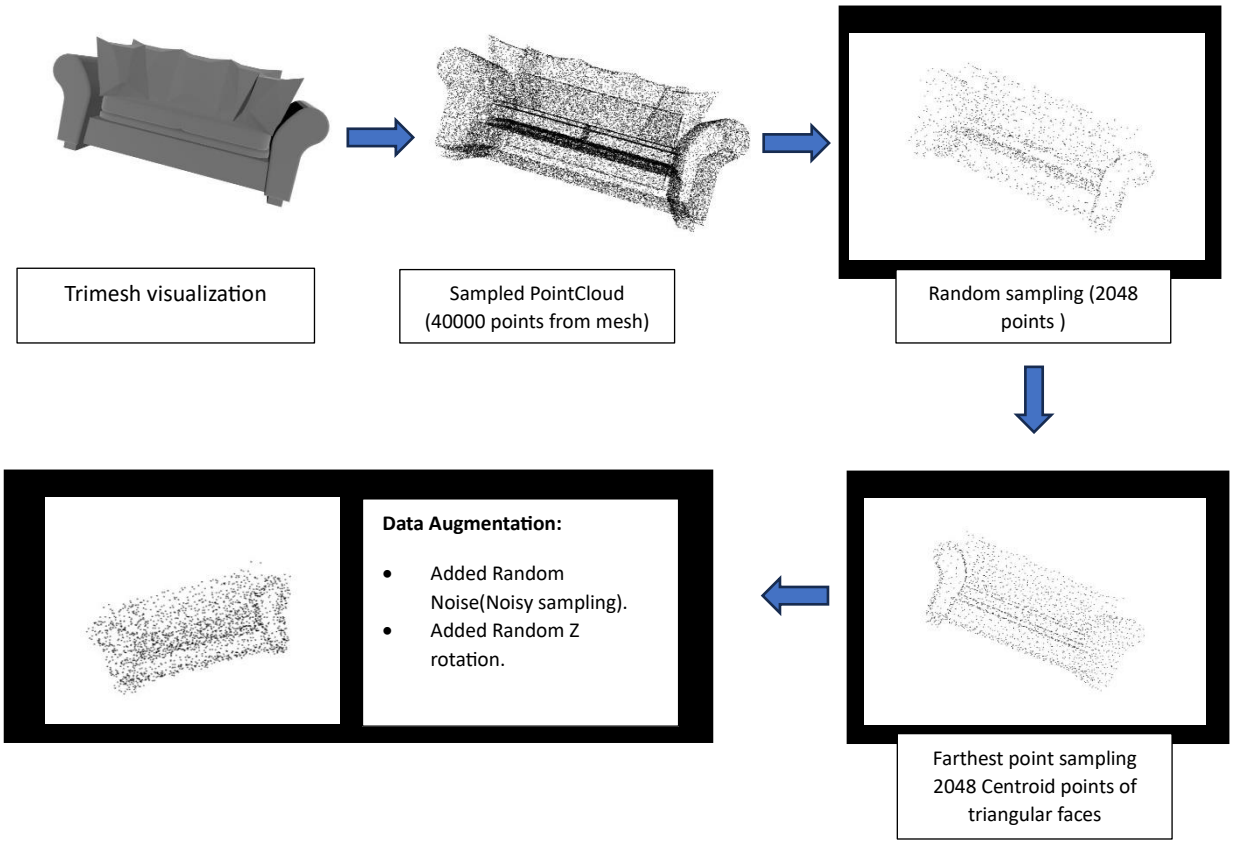
3991 models for training and 908 for testing.
Train dataset size: 3991
Valid dataset size: 908
Number of classes: 10

Data Visualization:

			
Open 3d plot	**Plotly – Mesh plot**	**Plotly – Point cloud plot**	**Plotly – Vertices plot**

**the codes above plots used ModelNet10/chair/train/chair_0003.off*

Visualization of Data Preprocessing:



**the codes of above plots used ModelNet10/sofa/train/sofa_0002.off*

Classification performance:

Following table illustrates the trained PointNet's performance in terms of accuracies on prediction over Test dataset.

Overall Accuracy(%)	Bathtub	Bed	Chair	Desk	Dresser	Monitor	Night stand	sofa	Table	Toilet
83.93	82	79	99	66	71	95	67	86	85	95

Setup Environment:

```
CUDA device: NVIDIA GeForce RTX 3080 Ti Laptop GPU
```

Code Performance:

The final prediction performance of trained model is as:

1. Number of epoch of training : 8
2. Number of batches per epoch : 125
3. Training Accuracy : 87.35 %
4. Validation Accuracy : **83.93 %**
5. training loss : 0.385
6. Validation Loss : 0.477

Loss Function definition:

To ensure stability with LogSoftmax, it's preferable to employ NLLLoss rather than CrossEntropyLoss or MSE Loss. Additionally, we'll incorporate two regularization terms to encourage transformation matrices to approach orthogonality (where $AA^T = I$).

1. **Classification Loss (NLLLoss):** The classification loss component is computed using the negative log likelihood loss (NLLLoss) between the predicted outputs and the ground truth labels:

$$\text{classification\_loss} = -\frac{1}{N}\sum_{i=1}^{N}\log\left(\text{softmax}(y_i)_{\text{true}}\right)$$

2. **Regularization Term for Transformation Matrices:** The regularization term penalizes the deviation of the transformation matrices from orthogonality:

$$\text{regularization\_term} = \alpha\left(\frac{1}{N}\sum_{i=1}^{N}\left\|\mathbf{I}_3 - \mathbf{M}_{3\times3}^{i}\cdot(\mathbf{M}_{3\times3}^{i})^{\mathrm{T}}\right\|_F + \frac{1}{N}\sum_{i=1}^{N}\left\|\mathbf{I}_{64} - \mathbf{M}_{64\times64}^{i}\cdot(\mathbf{M}_{64\times64}^{i})^{\mathrm{T}}\right\|_F\right)$$

   - α: Regularization coefficient.
   - N: Batch size.
   - $\mathbf{I}_3$ and $\mathbf{I}_{64}$: Identity matrices of sizes 3×3 and 64×64 respectively.
   - $\mathbf{M}_{3\times3}^{i}$ and $\mathbf{M}_{64\times64}^{i}$: Transformation matrices for 3×3 and 64×64 transformations respectively, corresponding to the i-th sample in the batch.
   - $\|\cdot\|_F$: Frobenius norm.

3. **Total Loss:** The total loss is the sum of the classification loss and the regularization term:

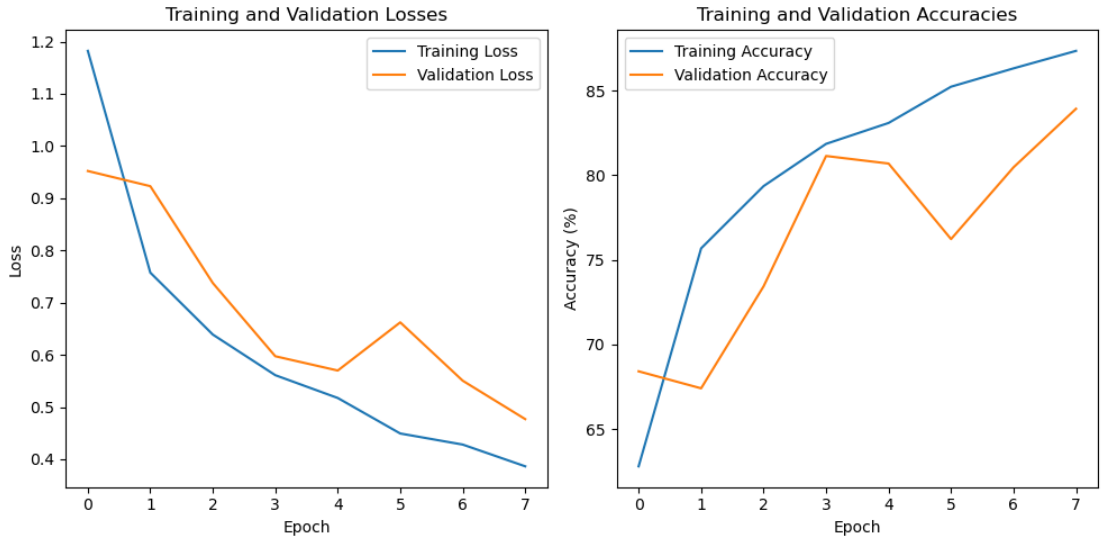
$$\text{total\_loss} = \text{classification\_loss} + \text{regularization\_term}.$$

Function definition as code:

```
def pointnetloss(outputs, labels, m3x3, m64x64, alpha = 0.0001):
    criterion = torch.nn.NLLLoss()  # Classification loss

    bs=outputs.size(0)
    id3x3 = torch.eye(3, requires_grad=True).repeat(bs,1,1)
    id64x64 = torch.eye(64, requires_grad=True).repeat(bs,1,1)
    if outputs.is_cuda:
        id3x3=id3x3.cuda()
        id64x64=id64x64.cuda()
    diff3x3 = id3x3-torch.bmm(m3x3,m3x3.transpose(1,2))
    diff64x64 = id64x64-torch.bmm(m64x64,m64x64.transpose(1,2))
    return criterion(outputs, labels) + alpha * (torch.norm(diff3x3)+torch.norm(diff64x64)) / float(bs)
```

Epoch Vs Losses curve and Epoch Vs Accuracies curve:



Model Architecture:

Following is the model architecture obtained from pytorch code.

PointNet(
  (transform): Transform(
    (input_transform): Tnet(
      (conv1): Conv1d(3, 64, kernel_size=(1,), stride=(1,))
      (conv2): Conv1d(64, 128, kernel_size=(1,), stride=(1,))
      (conv3): Conv1d(128, 1024, kernel_size=(1,), stride=(1,))
      (fc1): Linear(in_features=1024, out_features=512, bias=True)
      (fc2): Linear(in_features=512, out_features=256, bias=True)
      (fc3): Linear(in_features=256, out_features=9, bias=True)
      (bn1): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (bn2): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (bn3): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (bn4): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (bn5): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
)
    (feature_transform): Tnet(
      (conv1): Conv1d(64, 64, kernel_size=(1,), stride=(1,))
      (conv2): Conv1d(64, 128, kernel_size=(1,), stride=(1,))
      (conv3): Conv1d(128, 1024, kernel_size=(1,), stride=(1,))
      (fc1): Linear(in_features=1024, out_features=512, bias=True)
      (fc2): Linear(in_features=512, out_features=256, bias=True)

```
    (fc3): Linear(in_features=256, out_features=4096, bias=True)
    (bn1): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (bn2): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (bn3): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (bn4): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (bn5): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (conv1): Conv1d(3, 64, kernel_size=(1,), stride=(1,))
  (conv2): Conv1d(64, 128, kernel_size=(1,), stride=(1,))
  (conv3): Conv1d(128, 1024, kernel_size=(1,), stride=(1,))
  (bn1): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (bn2): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (bn3): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
 )
 (fc1): Linear(in_features=1024, out_features=512, bias=True)
 (fc2): Linear(in_features=512, out_features=256, bias=True)
 (fc3): Linear(in_features=256, out_features=10, bias=True)
 (bn1): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
 (bn2): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
 (dropout): Dropout(p=0.3, inplace=False)
 (logsoftmax): LogSoftmax(dim=1)
)

Layer Name: transform
Transform(
 (input_transform): Tnet(
  (conv1): Conv1d(3, 64, kernel_size=(1,), stride=(1,))
  (conv2): Conv1d(64, 128, kernel_size=(1,), stride=(1,))
  (conv3): Conv1d(128, 1024, kernel_size=(1,), stride=(1,))
  (fc1): Linear(in_features=1024, out_features=512, bias=True)
  (fc2): Linear(in_features=512, out_features=256, bias=True)
  (fc3): Linear(in_features=256, out_features=9, bias=True)
  (bn1): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (bn2): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (bn3): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (bn4): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (bn5): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
 )
 (feature_transform): Tnet(
  (conv1): Conv1d(64, 64, kernel_size=(1,), stride=(1,))
  (conv2): Conv1d(64, 128, kernel_size=(1,), stride=(1,))
  (conv3): Conv1d(128, 1024, kernel_size=(1,), stride=(1,))
  (fc1): Linear(in_features=1024, out_features=512, bias=True)
  (fc2): Linear(in_features=512, out_features=256, bias=True)
  (fc3): Linear(in_features=256, out_features=4096, bias=True)
  (bn1): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (bn2): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (bn3): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (bn4): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (bn5): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
 )
 (conv1): Conv1d(3, 64, kernel_size=(1,), stride=(1,))
 (conv2): Conv1d(64, 128, kernel_size=(1,), stride=(1,))
 (conv3): Conv1d(128, 1024, kernel_size=(1,), stride=(1,))
 (bn1): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
 (bn2): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
 (bn3): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
)
--------------------
Layer Name: fc1
```

Linear(in_features=1024, out_features=512, bias=True)

Layer Name: fc2
Linear(in_features=512, out_features=256, bias=True)

Layer Name: fc3
Linear(in_features=256, out_features=10, bias=True)

Layer Name: bn1
BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)

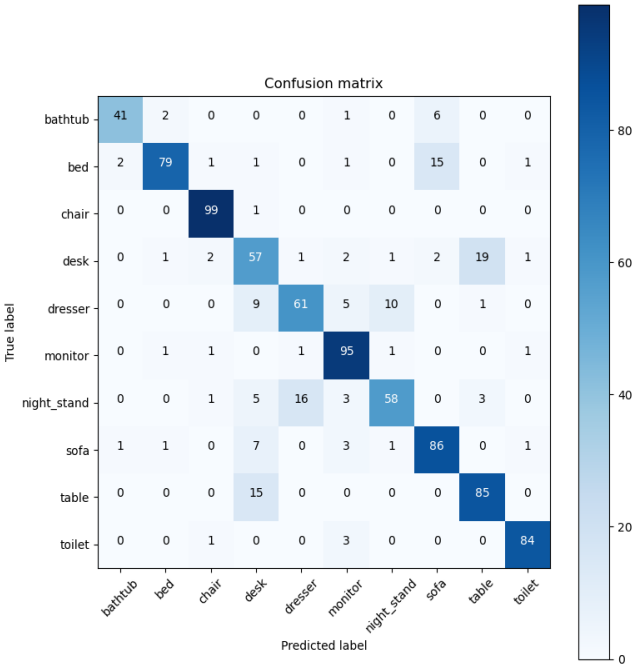
Layer Name: bn2
BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)

Layer Name: dropout
Dropout(p=0.3, inplace=False)

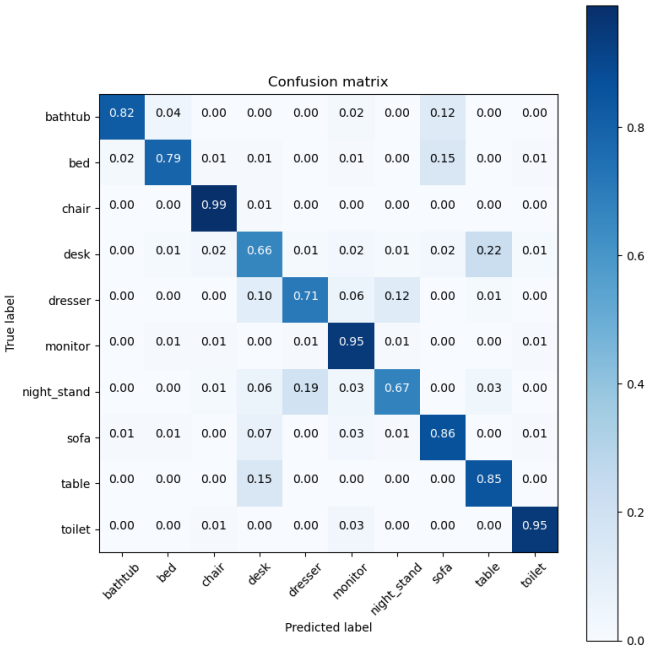
Layer Name: logsoftmax
LogSoftmax(dim=1)

Perfomance plots

Confusion Matrix Before Normalization:



Confusion Matrix After Normalization:

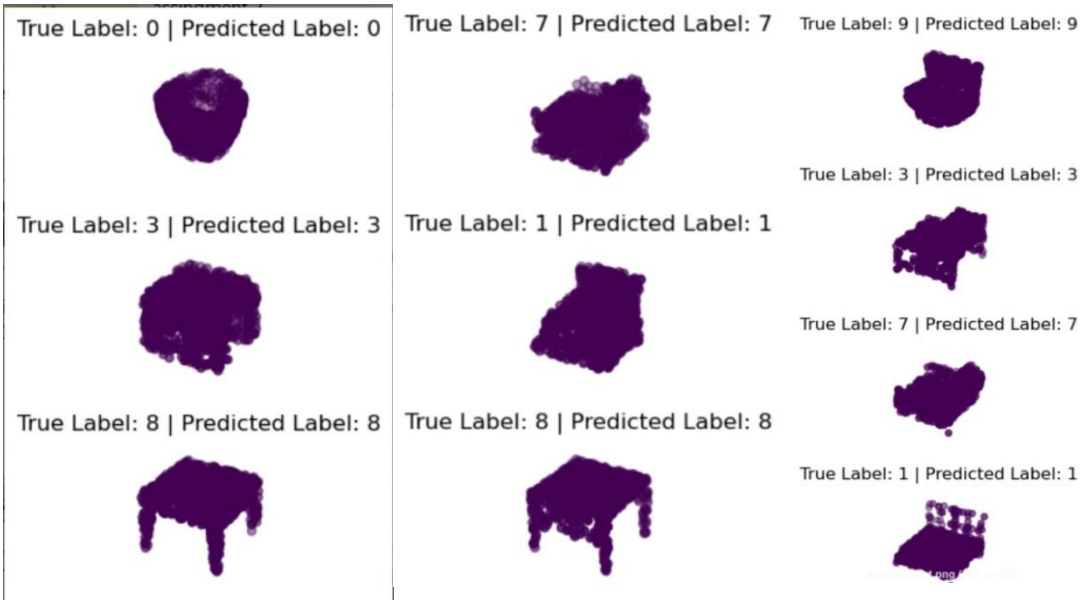


Classification performance:

Following table illustrates the trained PointNet's performance in terms of accuracies on prediction over Test dataset.

Overall Accuracy(%)	**Bathtub**	**Bed**	**Chair**	**Desk**	**Dresser**	**Monitor**	**Night stand**	**sofa**	**Table**	**Toilet**
83.93	82	79	99	66	71	95	67	86	85	95

Sample Predictions Vs Actual Labels



Description of attached files in Zip:

Main.ipynb	Contains the running code with cell by cell output.
Plots.ipynb	Contains various visualization methods used for PointCloud data that were utilized during data exploration process.
Model_architecture.png	The entire layer by layer Architecture diagram of the PointNet.
Readme.txt	Describes how to run the .ipynb files used to visualise and train point cloud
Implementation.pdf(This file)	Contains required output parameters and results.
model_acc83.pth	Saved model with 83+% of accuracy

Observation and Conclusion:

1. We have successfully implemented PointNet architecture for ModelNet10 object classification.
2. Achieved 83% accuracy in classifying 10 object categories.
3. PointNet effectively handles unordered point cloud data.
4. ModelNet10 dataset provided diverse 3D objects for training.
5. Future work may include exploring different architectures and data augmentation techniques.