3D Cloud Classification : PointNet apply to ModelNet10

In [21]:
```
#import all the necessary stuffs
import torch
import os
import numpy as np
from matplotlib import pyplot as plt
import random
import math
from torch.utils.data import Dataset, DataLoader, TensorDataset
from torchvision import transforms, utils
import torch.nn as nn
import torch.nn.functional as F
if torch.cuda.is_available():
  dev = "cuda:0"
else:
  dev = "cpu"
device = torch.device(dev)
global device
print(device)
```

```
cuda:0
```

In [22]:
```
from torch.utils.tensorboard import SummaryWriter

#
#
#      0===============================0
#      |    PLY files reader/writer    |
#      0===============================0
#
#
#----------------------------------------------------------------------------------
#
#      function to read/write .ply files
#
#----------------------------------------------------------------------------------
#
#      Hugues THOMAS - 10/02/2017
#


#----------------------------------------------------------------------------------
#
#          Imports and global variables
#      \**********************************/
#


# Basic libs
import numpy as np
import sys


# Define PLY types
ply_dtypes = dict([
    (b'int8', 'i1'),
    (b'char', 'i1'),
    (b'uint8', 'u1'),
    (b'uchar', 'b1'),
```

```python
    (b'uchar', 'u1'),
    (b'int16', 'i2'),
    (b'short', 'i2'),
    (b'uint16', 'u2'),
    (b'ushort', 'u2'),
    (b'int32', 'i4'),
    (b'int', 'i4'),
    (b'uint32', 'u4'),
    (b'uint', 'u4'),
    (b'float32', 'f4'),
    (b'float', 'f4'),
    (b'float64', 'f8'),
    (b'double', 'f8')
])

# Numpy reader format
valid_formats = {'ascii': '', 'binary_big_endian': '>',
                 'binary_little_endian': '<'}


#------------------------------------------------------------------------------------
#
#           Functions
#       \***************/
#

def parse_header(plyfile, ext):

    # Variables
    line = []
    properties = []
    num_points = None

    while b'end_header' not in line and line != b'':
        line = plyfile.readline()
        if b'element' in line:
            line = line.split()
            num_points = int(line[2])

        elif b'property' in line:
            line = line.split()
            properties.append((line[2].decode(), ext + ply_dtypes[line[1]]))

    return num_points, properties


def read_ply(filename):
    """
    Read ".ply" files

    Parameters
    ----------
    filename : string
        the name of the file to read.

    Returns
    -------
    result : array
        data stored in the file

    Examples
    --------
    Store data in file

    >>> points = np.random.rand(5, 3)
    >>> values = np.random.randint(2, size=10)
```

```python
    >>> write_ply('example.ply', [points, values], ['x', 'y', 'z', 'values'])

    Read the file

    >>> data = read_ply('example.ply')
    >>> values = data['values']
    array([0, 0, 1, 1, 0])

    >>> points = np.vstack((data['x'], data['y'], data['z'])).T
    array([[ 0.466  0.595  0.324]
           [ 0.538  0.407  0.654]
           [ 0.850  0.018  0.988]
           [ 0.395  0.394  0.363]
           [ 0.873  0.996  0.092]])

    """

    with open(filename, 'rb') as plyfile:
        # Check if the file start with ply
        if b'ply' not in plyfile.readline():
            raise ValueError('The file does not start whith the word ply')

        # get binary_little/big or ascii
        fmt = plyfile.readline().split()[1].decode()
        if fmt == "ascii":
            raise ValueError('The file is not binary')

        # get extension for building the numpy dtypes
        ext = valid_formats[fmt]

        # Parse header
        num_points, properties = parse_header(plyfile, ext)

        # Get data
        data = np.fromfile(plyfile, dtype=properties, count=num_points)
    return data


def header_properties(field_list, field_names):

    # List of lines to write
    lines = []

    # First line describing element vertex
    lines.append('element vertex %d' % field_list[0].shape[0])

    # Properties lines
    i = 0
    for fields in field_list:
        for field in fields.T:
            lines.append('property %s %s' % (field.dtype.name, field_names[i]))
            i += 1

    return lines


def write_ply(filename, field_list, field_names):
    """
    Write ".ply" files

    Parameters
    ----------
    filename : string
        the name of the file to which the data is saved. A '.ply' extension will
        be appended to the file name if it does no already have one.
```

```
    field_list : list, tuple, numpy array
        the fields to be saved in the ply file. Either a numpy array, a list of
        numpy arrays or a tuple of numpy arrays. Each 1D numpy array and each
        column of 2D numpy arrays are considered as one field.

    field_names : list
        the name of each fields as a list of strings. Has to be the same length
        as the number of fields.

    Examples
    --------
    >>> points = np.random.rand(10, 3)
    >>> write_ply('example1.ply', points, ['x', 'y', 'z'])

    >>> values = np.random.randint(2, size=10)
    >>> write_ply('example2.ply', [points, values], ['x', 'y', 'z', 'values'])

    >>> colors = np.random.randint(255, size=(10,3), dtype=np.uint8)
    >>> field_names = ['x', 'y', 'z', 'red', 'green', 'blue', values']
    >>> write_ply('example3.ply', [points, colors, values], field_names)

    """

    # Format list input to the right form
    field_list = list(field_list) if (type(field_list) == list or type(field_list) ==
    for i, field in enumerate(field_list):
        if field is None:
            print('WRITE_PLY ERROR: a field is None')
            return False
        elif field.ndim > 2:
            print('WRITE_PLY ERROR: a field have more than 2 dimensions')
            return False
        elif field.ndim < 2:
            field_list[i] = field.reshape(-1, 1)

    # check all fields have the same number of data
    n_points = [field.shape[0] for field in field_list]
    if not np.all(np.equal(n_points, n_points[0])):
        print('wrong field dimensions')
        return False

    # Check if field_names and field_list have same nb of column
    n_fields = np.sum([field.shape[1] for field in field_list])
    if (n_fields != len(field_names)):
        print('wrong number of field names')
        return False

    # Add extension if not there
    if not filename.endswith('.ply'):
        filename += '.ply'

    # open in text mode to write the header
    with open(filename, 'w') as plyfile:

        # First magical word
        header = ['ply']

        # Encoding format
        header.append('format binary_' + sys.byteorder + '_endian 1.0')

        # Points properties description
        header.extend(header_properties(field_list, field_names))

        # End of header
        header.append('end_header')
```

```
        # Write all lines
        for line in header:
            plyfile.write("%s\n" % line)

    # open in binary/append to use tofile
    with open(filename, 'ab') as plyfile:

        # Create a structured array
        i = 0
        type_list = []
        for fields in field_list:
            for field in fields.T:
                type_list += [(field_names[i], field.dtype.str)]
                i += 1
        data = np.empty(field_list[0].shape[0], dtype=type_list)
        i = 0
        for fields in field_list:
            for field in fields.T:
                data[field_names[i]] = field
                i += 1

        data.tofile(plyfile)

    return True


def describe_element(name, df):
    """ Takes the columns of the dataframe and builds a ply-like description

    Parameters
    ----------
    name: str
    df: pandas DataFrame

    Returns
    -------
    element: list[str]
    """
    property_formats = {'f': 'float', 'u': 'uchar', 'i': 'int'}
    element = ['element ' + name + ' ' + str(len(df))]

    if name == 'face':
        element.append("property list uchar int points_indices")

    else:
        for i in range(len(df.columns)):
            # get first letter of dtype to infer format
            f = property_formats[str(df.dtypes[i])[0]]
            element.append('property ' + f + ' ' + df.columns.values[i])

    return element
```

In [23]:

```
#define default and custom transformation for 3D cloud objects

class ToTensor(object):
    def __call__(self, pointcloud):
        return torch.from_numpy(pointcloud).to(torch.float32).to(device)

class RandomRotation_z(object):
    def __call__(self, pointcloud):

        theta = random.random() * 2. * math.pi
        rot_matrix = torch.tensor([[math.cos(theta), -math.sin(theta),      0],
                                   [math.sin(theta),  math.cos(theta),      0],
                                   [0,                               0,      1]],dtype=tor
```

```
        rot_pointcloud = torch.matmul(pointcloud,rot_matrix)
        return rot_pointcloud

class RandomNoise(object):
    def __call__(self, pointcloud):

        noise = torch.rand(pointcloud.size(0),pointcloud.size(1)).to(device)*0.02
        noisy_pointcloud = pointcloud + noise
        return noisy_pointcloud

class ShufflePoints(object):
    def __call__(self, pointcloud):
        index = torch.randperm(pointcloud.size(0))
        pointcloud[:] = pointcloud[index]
        return pointcloud

class AxisReducer(object):
    def __call__(self, pointcloud):
        pointcloud[:,0] = torch.sqrt(torch.square(pointcloud[:,0]) + torch.square(poi
        pointcloud[:,1] = 0
        return pointcloud

class NormalizePoints(object):
    def __call__(self, pointcloud):
        return pointcloud/(torch.max(torch.min(pointcloud),torch.max(pointcloud)))

class PointsToVoxel(object):
    def __call__(self, pointcloud,voxel_size = 8):
        pointcloud= (((pointcloud+1)/2.01)*voxel_size).int()
        return pointcloud

class VoxelToBool(object): #to visualize the result of PointsToVoxel
    def __call__(self, pointcloud,voxel_size = 8):
        bool_array = torch.zeros((voxel_size,voxel_size,voxel_size,),dtype=bool)
        for i in range(pointcloud.size(0)):
            bool_array[pointcloud[i][0],pointcloud[i][1],pointcloud[i][2]]=True
        return bool_array

def default_transforms():
    return transforms.Compose([
        ToTensor(),
        RandomRotation_z(),
        RandomNoise(),
        ShufflePoints(),
        ])

def customize_transforms():
    return transforms.Compose([
        ToTensor(),
        RandomRotation_z(),
        RandomNoise(),
        AxisReducer(),
        ShufflePoints(),
        ])

def customize_transforms_voxel():
    return transforms.Compose([
        ToTensor(),
        RandomRotation_z(),
        RandomNoise(),
        NormalizePoints(),
        PointsToVoxel(),
        ShufflePoints(),
        ])
```

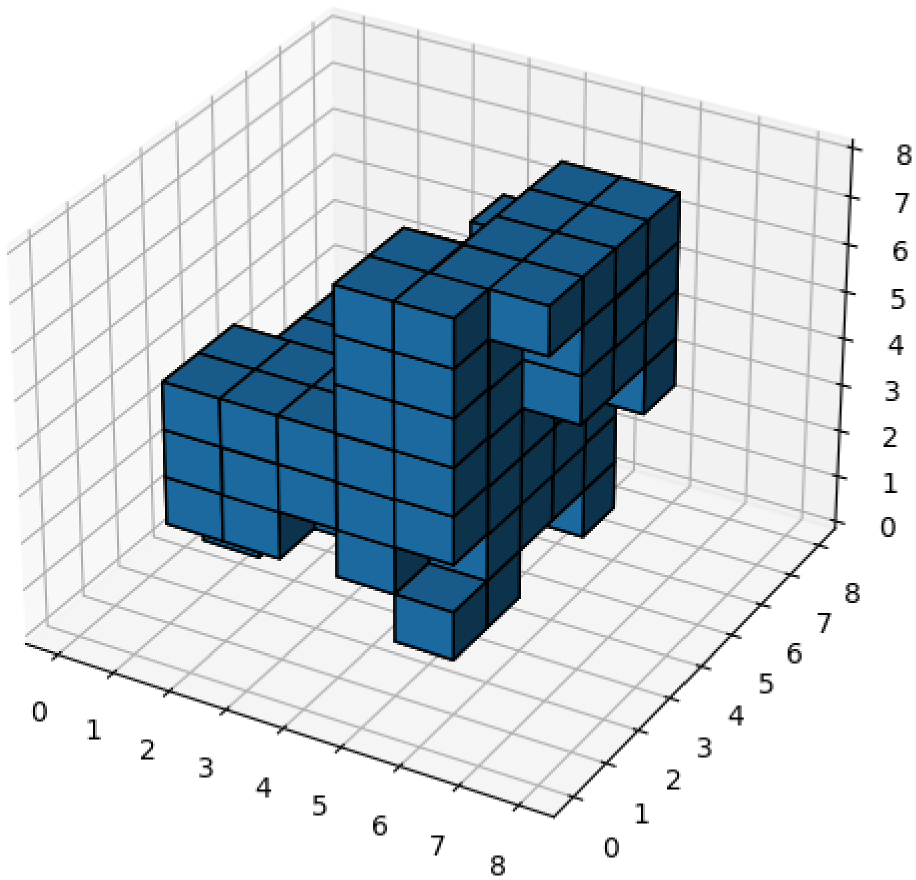
In [24]:

```python
#define our and verify our dataset
class PointCloudData(Dataset):
    def __init__(self,
                 root_dir,
                 folder="train",
                 transform=default_transforms()):
        self.root_dir = root_dir
        folders = [dir for dir in sorted(os.listdir(root_dir))
                   if os.path.isdir(root_dir + "/" + dir)]
        self.classes = {folder: i for i, folder in enumerate(folders)}
        self.transforms = transform
        self.files = []
        for category in self.classes.keys():
            new_dir = root_dir+"/"+category+"/"+folder
            for file in os.listdir(new_dir):
                if file.endswith('.ply'):
                    sample = {}
                    sample['ply_path'] = new_dir+"/"+file
                    sample['category'] = category
                    self.files.append(sample)

    def __len__(self):
        return len(self.files)

    def __getitem__(self, idx):
        ply_path = self.files[idx]['ply_path']
        category = self.files[idx]['category']
        data = read_ply(ply_path)
        pointcloud = self.transforms(np.vstack((data['x'],
                                                data['y'],
                                                data['z'])).T)
        return {'pointcloud': pointcloud, 'category': self.classes[category]}


def slice_dataset(dataset_input):
    dataset = []
    for i in range(10):
        dataset.append([])

    for obj in dataset_input:
        index = obj["category"]
        dataset[index].append(obj["pointcloud"])
    torch_dataset = []
    for data in dataset:
        #print(torch.cat(data))

        torch_dataset.append(torch.stack(data))

    return torch_dataset



NUM_POINTS = 1024
NUM_CLASSES = 10
BATCH_SIZE = 32


train_ds = PointCloudData("./data/ModelNet10_PLY")
test_ds = PointCloudData("./data/ModelNet10_PLY", folder='test')


dataloader_train = DataLoader(train_ds, batch_size=BATCH_SIZE, shuffle=True)
dataloader_test = DataLoader(test_ds, batch_size=BATCH_SIZE, shuffle=True)
```

```
train_ds_augment = PointCloudData("./data/ModelNet10_PLY",transform=customize_transfo
test_ds_augment = PointCloudData("./data/ModelNet10_PLY", folder='test',transform=cus

dataloader_train_augment = DataLoader(train_ds_augment, batch_size=BATCH_SIZE, shuffl
dataloader_test_augment = DataLoader(test_ds_augment, batch_size=BATCH_SIZE, shuffle=

train_ds_augment_voxel = PointCloudData("./data/ModelNet10_PLY",transform=customize_t
test_ds_augment_voxel = PointCloudData("./data/ModelNet10_PLY", folder='test',transfo

dataloader_train_augment_voxel = DataLoader(train_ds_augment_voxel, batch_size=BATCH_
dataloader_test_augment_voxel = DataLoader(test_ds_augment_voxel, batch_size=BATCH_SI




index =800


plt.figure(1)
points = train_ds[index]["pointcloud"].cpu()


plt.figure(1)
fig = plt.figure(figsize=(5, 5))
ax = fig.add_subplot(111, projection="3d")
ax.scatter(points[:, 0], points[:, 1], points[:, 2],s=5)
ax.set_axis_off()
plt.show()



points = train_ds_augment_voxel[index]["pointcloud"].cpu()

voxel_to_bool = VoxelToBool()

voxelarray = voxel_to_bool(points)
print(voxelarray.shape)


plt.rcParams["figure.figsize"] = [5,5]
plt.rcParams["figure.autolayout"] = True
ax = plt.figure(2).add_subplot(projection='3d')



#ax.voxels(voxelarray, edgecolor="k",facecolors="red")
ax.voxels(voxelarray, edgecolor='k')

plt.show()
print(voxelarray.shape)
```

```
<Figure size 640x480 with 0 Axes>
```

```
torch.Size([8, 8, 8])
/home/a/.local/lib/python3.8/site-packages/IPython/core/pylabtools.py:152: UserWarnin
g: This figure includes Axes that are not compatible with tight_layout, so results mi
ght be incorrect.
  fig.canvas.print_figure(bytes_io, **kw)
```

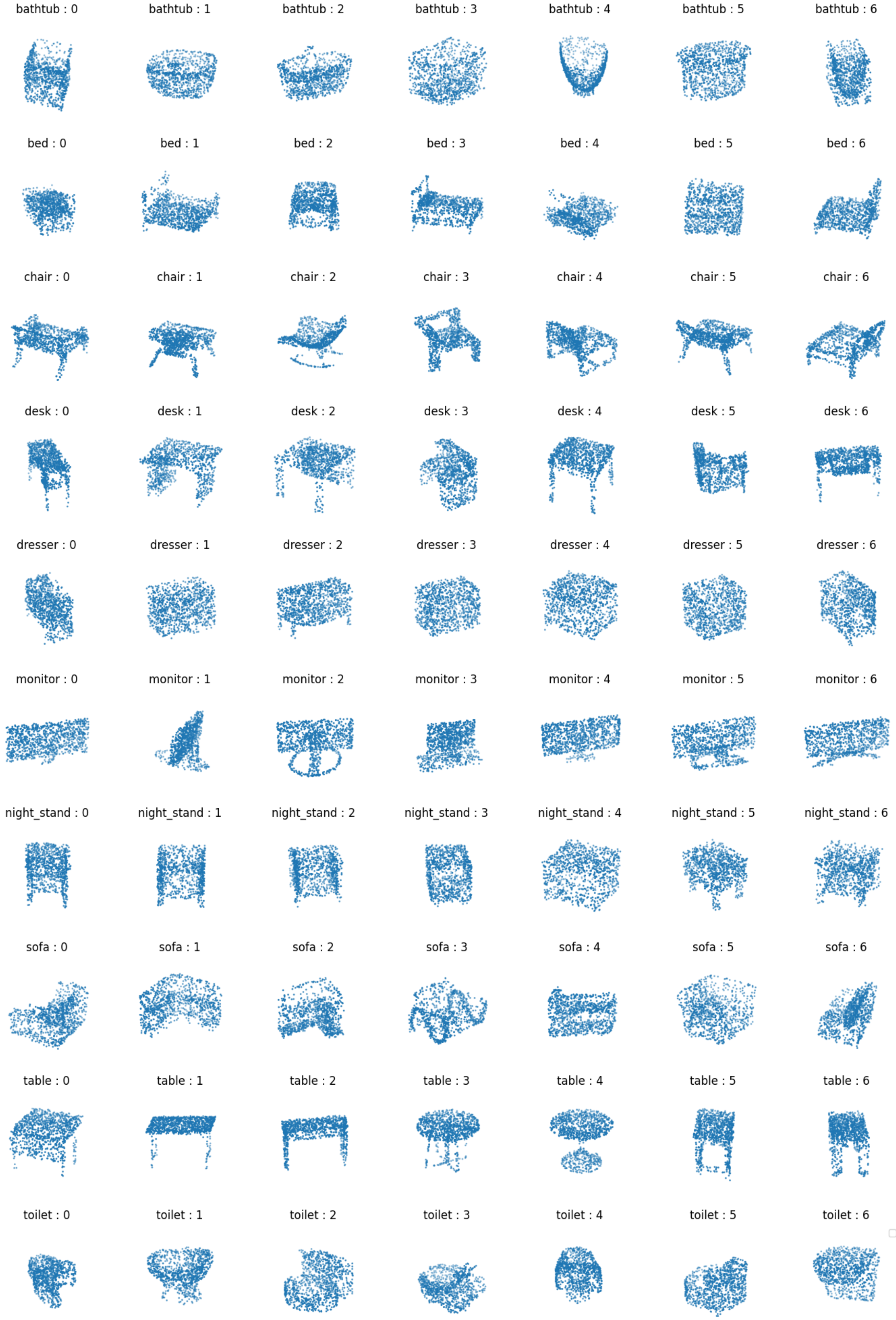



```
torch.Size([8, 8, 8])
```

In [25]:
```
# show an example of each object
num_classes = 10
titles_object =[]
for key in train_ds_augment.classes:
    titles_object.append(key)
```

```
# fig, ax = plt.subplots(num_classes, 5, figsize=(10,20))
fig = plt.figure(figsize=(14,20))

sliced_dataset = slice_dataset(train_ds)
sliced_dataset_test = slice_dataset(test_ds)

size_x = 7
for i in range(num_classes):
    for j in range(size_x):
        points = sliced_dataset[i][j].cpu()

        ax = fig.add_subplot(num_classes, size_x, 1+i*size_x+j, projection='3d')
        plt.title(titles_object[i]+" : "+str(j))

        ax.scatter(points[:, 0], points[:, 1], points[:, 2], s=1)


        ax.set_axis_off()

plt.legend()
plt.show()

print("Numbers of each object")
for i in range(num_classes):
    print(i,len(sliced_dataset[i]),len(sliced_dataset_test[i]))

del sliced_dataset,sliced_dataset_test
```

```
No artists with labels found to put in legend.  Note that artists whose label start w
ith an underscore are ignored when legend() is called with no argument.
```



bathtub : 0
bathtub : 1
bathtub : 2
bathtub : 3
bathtub : 4
bathtub : 5
bathtub : 6
bed : 0
bed : 1
bed : 2
bed : 3
bed : 4
bed : 5
bed : 6
chair : 0
chair : 1
chair : 2
chair : 3
chair : 4
chair : 5
chair : 6
desk : 0
desk : 1
desk : 2
desk : 3
desk : 4
desk : 5
desk : 6
dresser : 0
dresser : 1
dresser : 2
dresser : 3
dresser : 4
dresser : 5
dresser : 6
monitor : 0
monitor : 1
monitor : 2
monitor : 3
monitor : 4
monitor : 5
monitor : 6
night_stand : 0
night_stand : 1
night_stand : 2
night_stand : 3
night_stand : 4
night_stand : 5
night_stand : 6
sofa : 0
sofa : 1
sofa : 2
sofa : 3
sofa : 4
sofa : 5
sofa : 6
table : 0
table : 1
table : 2
table : 3
table : 4
table : 5
table : 6
toilet : 0
toilet : 1
toilet : 2
toilet : 3
toilet : 4
toilet : 5
toilet : 6

```
Numbers of each object
0 106 50
1 515 100
2 889 100
3 200 86
4 200 86
5 465 100
6 200 86
7 680 100
8 392 100
9 344 100
```

In [26]:

```python
#define plot, loss and train loop of our dataset
def plot_all(accuracy_train_array,accuracy_test_array,loss_train_array,loss_test_arra
    plt.figure(1)
    plt.xlabel("Epochs")
    plt.ylabel("Accuracy in %")
    plt.plot(accuracy_train_array,label="Accuracy train")
    plt.plot(accuracy_test_array,label="Accuracy test")

    plt.legend()
    plt.show()
    plt.figure(2)
    plt.xlabel("Epochs")
    plt.ylabel("Loss")

    plt.plot(loss_train_array,label="Loss train")
    plt.plot(loss_test_array,label="Loss test")

    plt.legend()
    plt.show()
def basic_loss(outputs, labels):
    #outputs,labels =  torch.Tensor(outputs,dtype=torch.long),torch.Tensor(labels,dty
    #criterion = torch.nn.NLLLoss()
    criterion = torch.nn.CrossEntropyLoss()
    #criterion = torch.nn.CrossEntropyLoss()
    bsize = outputs.size(0)
    #outputs = torch.transpose(outputs,0, 1)
    return criterion(outputs, labels)


def pointnet_full_loss(outputs, labels, m1, m2, loss_func, alpha=0.001):
    #criterion = torch.nn.NLLLoss()
    criterion = loss_func
    #criterion = torch.nn.CrossEntropyLoss()
    bsize = outputs.size(0)


    id_1 = torch.eye(m1.size(1), requires_grad=True).repeat(bsize, 1, 1).to(device)
    diff1 = id_1 - torch.bmm(m1, m1.transpose(1, 2))

    id_2 = torch.eye(m2.size(1), requires_grad=True).repeat(bsize, 1, 1).to(device)
    diff2 = id_2 - torch.bmm(m2, m2.transpose(1, 2))


    return criterion(outputs, labels) + alpha * (torch.norm(diff1)) / float(bsize) +

def get_accuracy(labels_predict,labels_true):
    _, predicted = torch.max(labels_predict.data, 1)
    total = labels_true.size(0)
    correct = (predicted == labels_true).sum().item()
    val_acc = 100. * correct / total
    return val_acc

def evaluation_model(model,dataloader_test,loss_func):
    correct = total = 0
```

```
    loss_test=0
    val_acc_test=0
    size=0
    with torch.no_grad():
        for id_batch, data in enumerate(dataloader_test):
            inputs, labels = data['pointcloud'].float(), data['category']
            size+=1
            predicted,rotation_1,rotation_2 = model(inputs)
                        # outputs, __ = model(inputs.transpose(1,2))

            predicted, labels = torch.Tensor(predicted).type(torch.FloatTensor),torch
            #loss_test = basic_loss(predicted, labels)
            loss_test += pointnet_full_loss(predicted, labels,rotation_1,rotation_2,l
            val_acc_test += get_accuracy(predicted,labels)

    return loss_test/size,val_acc_test/size


def train(
    model,
    dataloader_train,
    dataloader_test,
    epochs=100,
    loss_func=torch.nn.NLLLoss()
):
    optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
    scheduler = torch.optim.lr_scheduler.StepLR(optimizer,
                                                step_size=20, gamma=0.5)

    (loss_train_array,accuracy_train_array,loss_test_array,accuracy_test_array,)= [],

    for epoch in range(epochs):

        for id_batch, data in enumerate(dataloader_train):
            inputs, labels = data['pointcloud'].float(), data['category']


            optimizer.zero_grad()
            labels_predict,rotation_1,rotation_2 = model(inputs)
            labels_predict, labels = torch.Tensor(labels_predict).type(torch.FloatTen
            #loss_train = basic_loss(labels_predict, labels)

            loss_train = pointnet_full_loss(labels_predict, labels,rotation_1,rotatio

            loss_train.backward()
            optimizer.step()
            #print(id_batch)
            if id_batch==0:

                acc_train = get_accuracy(labels_predict,labels)
                loss_train_array.append(loss_train.cpu().detach().numpy())
                accuracy_train_array.append(acc_train)
                scheduler.step()
                loss_test,acc_test=evaluation_model(model,dataloader_test,loss_func)
                loss_test_array.append(loss_test.cpu().detach().numpy())
                accuracy_test_array.append(acc_test)

                print('Epoch: %d, Loss_train: %.3f, Accuracy_train: %.1f %%, Loss_tes

    return loss_train_array,accuracy_train_array,loss_test_array,accuracy_test_array
```

In [27]:

```
#define our different model

class PointMLP(nn.Module):
    def __init__(self, input_size, classes=10):
```

```python
        super(PointMLP, self).__init__()
        l_1 = 512
        l_2 = 256

        self.fc_1 = nn.Linear(NUM_POINTS*3,l_1).to(device)
        self.bn_1 = nn.BatchNorm1d(l_1).to(device)
        self.fc_2 = nn.Linear(l_1,l_2).to(device)
        self.dropout_1 = nn.Dropout(0.3).to(device)
        self.bn_2 = nn.BatchNorm1d(l_2).to(device)

        self.fc_3 = nn.Linear(l_2,classes).to(device)
        self.bn_3 = nn.BatchNorm1d(classes).to(device)

        self.eye_1 = torch.eye(1, requires_grad=False)
        self.eye_2 = torch.eye(1, requires_grad=False)


    def forward(self, x):
        x = x.to(device)

        x = torch.flatten(x,start_dim=1)
        x = self.fc_1(x)
        x = self.bn_1(x)
        x = F.relu(x)
        x = self.fc_2(x)
        x = self.dropout_1(x)
        x = self.bn_2(x)
        x = F.relu(x)
        x = self.fc_3(x)
        x = self.bn_3(x)
        x = F.relu(x)
        return x,self.eye_1.repeat(x.size(0), 1, 1).to(device),self.eye_2.repeat(x.si


class Tnet(nn.Module):
    def __init__(self, input_size, kernel_size):
        super(Tnet, self).__init__()
        #l_1 = 64
        #l_2 = 128
        #l_3 = 1024
        #l_4 = 512
        #l_5 = 256

        l_1 = 32
        l_2 = 64
        l_3 = 256
        l_4 = 128
        l_5 = 64

        self.kn_size = kernel_size

        self.fc_1 = nn.Conv1d(kernel_size,l_1,1).to(device)
        self.bn_1 = nn.BatchNorm1d(l_1).to(device)
        self.fc_2 = nn.Conv1d(l_1,l_2,1).to(device)
        self.bn_2 = nn.BatchNorm1d(l_2).to(device)
        self.fc_3 = nn.Conv1d(l_2,l_3,1).to(device)
        self.bn_3 = nn.BatchNorm1d(l_3).to(device)

        self.mp = nn.MaxPool1d(l_3).to(device)
        self.fc_4 = nn.Linear(input_size,l_4).to(device)
        self.bn_4 = nn.BatchNorm1d(l_4).to(device)
        self.fc_5 = nn.Linear(l_4,l_5).to(device)
        self.bn_5 = nn.BatchNorm1d(l_5).to(device)
        self.fc_6 = nn.Linear(l_5,self.kn_size*self.kn_size).to(device)

    def forward(self, x):
```

```
        x = x.to(device)

        #x = torch.flatten(x,start_dim=-2)
        x = x.transpose(2, 1)
        x = self.fc_1(x)
        x = self.bn_1(x)
        x = F.relu(x)
        x = self.fc_2(x)
        x = self.bn_2(x)
        x = F.relu(x)
        x = self.fc_3(x)
        x = self.bn_3(x)
        x = F.relu(x)

        x = self.mp(x)
        x = torch.flatten(x,start_dim=1)
        x = self.fc_4(x)
        x = self.bn_4(x)
        x = F.relu(x)
        x = self.fc_5(x)
        x = self.bn_5(x)
        x = F.relu(x)
        x = self.fc_6(x)
        x = x.view(-1,self.kn_size,self.kn_size)
        return x

class InputTransform(nn.Module):
    def __init__(self,input_size, kernel_size):
        super(InputTransform, self).__init__()
        self.kn_size = kernel_size

        self.t_net = Tnet(input_size, kernel_size).to(device)

    def forward(self, x):
        x = x.to(device)
        kern = self.t_net(x)
        x = torch.matmul(x,kern)
        return x,kern


class PointNetBasic(nn.Module):
    def __init__(self, input_size,classes=10):
        super(PointNetBasic, self).__init__()
        #l_1 = 64
        #l_2 = 64
        #l_3 = 64
        #l_4 = 128
        #l_5 = 1024
        #l_6 = 512
        #l_7 = 256

        l_1 = 32
        l_2 = 32
        l_3 = 32
        l_4 = 128
        l_5 = 512
        l_6 = 256
        l_7 = 128

        self.fc_1 = nn.Conv1d(3,l_1,1).to(device)
        self.bn_1 = nn.BatchNorm1d(l_1).to(device)
        self.fc_2 = nn.Conv1d(l_1,l_2,1).to(device)
        self.bn_2 = nn.BatchNorm1d(l_2).to(device)
        self.fc_3 = nn.Conv1d(l_2,l_3,1).to(device)
        self.bn_3 = nn.BatchNorm1d(l_3).to(device)
        self.fc_4 = nn.Conv1d(l_3,l_4,1).to(device)
```

```
        self.bn_4 = nn.BatchNorm1d(l_4).to(device)
        self.fc_5 = nn.Conv1d(l_4,l_5,1).to(device)
        self.bn_5 = nn.BatchNorm1d(l_5).to(device)

        self.mp = nn.MaxPool1d(l_5).to(device)
        self.fc_6 = nn.Linear(input_size,l_6).to(device)
        self.bn_6 = nn.BatchNorm1d(l_6).to(device)
        self.fc_7 = nn.Linear(l_6,l_7).to(device)
        self.dropout_1 = nn.Dropout(0.3).to(device)
        self.bn_7 = nn.BatchNorm1d(l_7).to(device)
        self.fc_8 = nn.Linear(l_7,classes).to(device)
        self.eye_1 = torch.eye(1, requires_grad=False)
        self.eye_2 = torch.eye(1, requires_grad=False)

    def forward(self, x):
        x=x.to(device)
        x = x.transpose(2, 1)
        x = self.fc_1(x)
        x = self.bn_1(x)
        x = F.relu(x)
        x = self.fc_2(x)
        x = self.bn_2(x)
        x = F.relu(x)
        x = self.fc_3(x)
        x = self.bn_3(x)
        x = F.relu(x)
        x = self.fc_4(x)
        x = self.bn_4(x)
        x = F.relu(x)
        x = self.fc_5(x)
        x = self.bn_5(x)
        x = F.relu(x)

        x = self.mp(x)
        x = torch.flatten(x,start_dim=1)

        x = self.fc_6(x)
        x = self.bn_6(x)
        x = F.relu(x)
        x = self.fc_7(x)
        x = self.bn_7(x)
        x = F.relu(x)
        x = self.dropout_1(x)
        x = self.fc_8(x)
        return x,self.eye_1.repeat(x.size(0), 1, 1).to(device),self.eye_2.repeat(x.si



class PointNetFull(nn.Module):
    def __init__(self, input_size,classes=10):
        super(PointNetFull, self).__init__()
        #l_1 = 64
        #l_2 = 64
        #l_3 = 64
        #l_4 = 128
        #l_5 = 1024
        #l_6 = 512
        #l_7 = 256

        l_1 = 32
        l_2 = 32
        l_3 = 32
        l_4 = 128
        l_5 = 512
        l_6 = 256
```

```
        l_7 = 128

        #torch.nn.Conv1d(3, 64, 1)
        self.input_transform_1 = InputTransform(input_size,3).to(device)

        self.fc_1 = nn.Conv1d(3,l_1,1).to(device)
        self.bn_1 = nn.BatchNorm1d(l_1).to(device)
        self.fc_2 = nn.Conv1d(l_1,l_2,1).to(device)
        self.bn_2 = nn.BatchNorm1d(l_2).to(device)

        self.fc_3 = nn.Conv1d(l_2,l_3,1).to(device)
        self.bn_3 = nn.BatchNorm1d(l_3).to(device)
        self.fc_4 = nn.Conv1d(l_3,l_4,1).to(device)
        self.bn_4 = nn.BatchNorm1d(l_4).to(device)
        self.fc_5 = nn.Conv1d(l_4,l_5,1).to(device)
        self.bn_5 = nn.BatchNorm1d(l_5).to(device)

        self.mp = nn.MaxPool1d(l_5).to(device)
        self.fc_6 = nn.Linear(input_size,l_6).to(device)
        self.bn_6 = nn.BatchNorm1d(l_6).to(device)
        self.fc_7 = nn.Linear(l_6,l_7).to(device)
        self.dropout_1 = nn.Dropout(0.3).to(device)
        self.bn_7 = nn.BatchNorm1d(l_7).to(device)
        self.fc_8 = nn.Linear(l_7,classes).to(device)
        self.eye_1 = torch.eye(1, requires_grad=False)
        self.eye_2 = torch.eye(1, requires_grad=False)

    def forward(self, x):
        x=x.to(device)
        x,rotation_1 = self.input_transform_1(x)

        x = x.transpose(2, 1)
        x = self.fc_1(x)
        x = self.bn_1(x)
        x = F.relu(x)

        x = self.fc_2(x)
        x = self.bn_2(x)
        x = F.relu(x)

        x = self.fc_3(x)
        x = self.bn_3(x)
        x = F.relu(x)

        x = self.fc_4(x)
        x = self.bn_4(x)
        x = F.relu(x)

        x = self.fc_5(x)
        x = self.bn_5(x)
        x = F.relu(x)

        x = self.mp(x)
        x = torch.flatten(x,start_dim=1)

        x = self.fc_6(x)
        x = self.bn_6(x)
        x = F.relu(x)

        x = self.fc_7(x)
        x = self.bn_7(x)
        x = F.relu(x)
        x = self.dropout_1(x)
        x = self.fc_8(x)
        return x,rotation_1,self.eye_1.repeat(x.size(0), 1, 1).to(device)
```

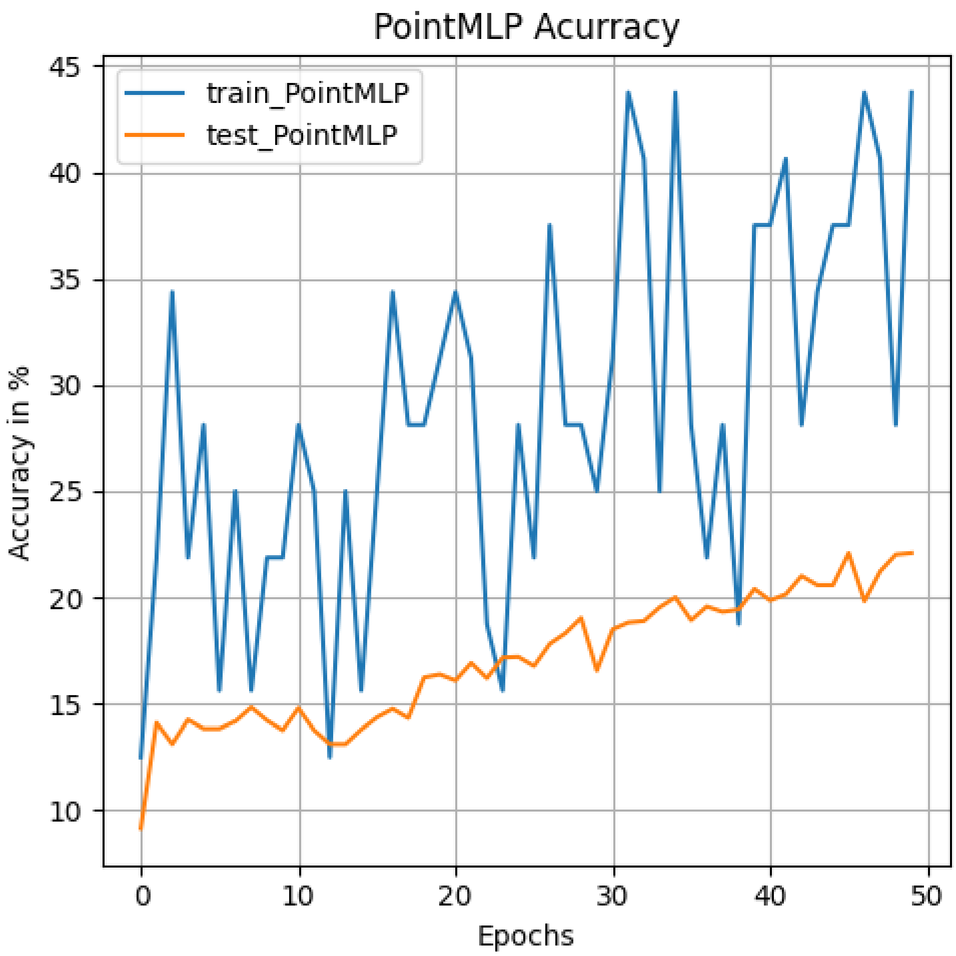
Results :

In [28]:
```
#PointMLP, no augment
model = PointMLP(NUM_POINTS,NUM_CLASSES)
(loss_train_array_POINTMLP,
 accuracy_train_array_POINTMLP,
 loss_test_array_POINTMLP,
 accuracy_test_array_POINTMLP) = train(
    model,
    dataloader_train,
    dataloader_test,
    epochs=50,
    loss_func=torch.nn.CrossEntropyLoss()
)

plt.figure(1)
plt.title("PointMLP Acurracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy in %")

plt.plot(accuracy_train_array_POINTMLP,label="train_PointMLP")
plt.plot(accuracy_test_array_POINTMLP,label="test_PointMLP")
plt.grid()
plt.legend()
```

```
Epoch: 1, Loss_train: 2.422, Accuracy_train: 12.5 %, Loss_test 2.495, Accuracy_test:
9.2 %
Epoch: 2, Loss_train: 2.261, Accuracy_train: 21.9 %, Loss_test 2.359, Accuracy_test:
14.1 %
Epoch: 3, Loss_train: 2.132, Accuracy_train: 34.4 %, Loss_test 2.328, Accuracy_test:
13.1 %
Epoch: 4, Loss_train: 2.119, Accuracy_train: 21.9 %, Loss_test 2.307, Accuracy_test:
14.3 %
Epoch: 5, Loss_train: 2.143, Accuracy_train: 28.1 %, Loss_test 2.303, Accuracy_test:
13.8 %
Epoch: 6, Loss_train: 2.306, Accuracy_train: 15.6 %, Loss_test 2.315, Accuracy_test:
13.8 %
Epoch: 7, Loss_train: 2.111, Accuracy_train: 25.0 %, Loss_test 2.286, Accuracy_test:
14.2 %
Epoch: 8, Loss_train: 2.206, Accuracy_train: 15.6 %, Loss_test 2.302, Accuracy_test:
14.8 %
Epoch: 9, Loss_train: 2.192, Accuracy_train: 21.9 %, Loss_test 2.305, Accuracy_test:
14.2 %
Epoch: 10, Loss_train: 2.216, Accuracy_train: 21.9 %, Loss_test 2.326, Accuracy_test:
13.8 %
Epoch: 11, Loss_train: 2.069, Accuracy_train: 28.1 %, Loss_test 2.298, Accuracy_test:
14.8 %
Epoch: 12, Loss_train: 2.110, Accuracy_train: 25.0 %, Loss_test 2.315, Accuracy_test:
13.8 %
Epoch: 13, Loss_train: 2.233, Accuracy_train: 12.5 %, Loss_test 2.316, Accuracy_test:
13.1 %
Epoch: 14, Loss_train: 2.141, Accuracy_train: 25.0 %, Loss_test 2.322, Accuracy_test:
13.1 %
Epoch: 15, Loss_train: 2.306, Accuracy_train: 15.6 %, Loss_test 2.335, Accuracy_test:
13.8 %
Epoch: 16, Loss_train: 2.090, Accuracy_train: 25.0 %, Loss_test 2.333, Accuracy_test:
14.4 %
Epoch: 17, Loss_train: 1.942, Accuracy_train: 34.4 %, Loss_test 2.313, Accuracy_test:
14.8 %
Epoch: 18, Loss_train: 1.930, Accuracy_train: 28.1 %, Loss_test 2.306, Accuracy_test:
14.3 %
Epoch: 19, Loss_train: 2.069, Accuracy_train: 28.1 %, Loss_test 2.314, Accuracy_test:
16.2 %
Epoch: 20, Loss_train: 1.989, Accuracy_train: 31.2 %, Loss_test 2.286, Accuracy_test:
16.4 %
Epoch: 21, Loss_train: 1.838, Accuracy_train: 34.4 %, Loss_test 2.283, Accuracy_test:
16.1 %
Epoch: 22, Loss_train: 2.143, Accuracy_train: 31.2 %, Loss_test 2.293, Accuracy_test:
16.9 %
Epoch: 23, Loss_train: 2.208, Accuracy_train: 18.8 %, Loss_test 2.286, Accuracy_test:
16.2 %
Epoch: 24, Loss_train: 2.044, Accuracy_train: 15.6 %, Loss_test 2.270, Accuracy_test:
17.2 %
Epoch: 25, Loss_train: 2.129, Accuracy_train: 28.1 %, Loss_test 2.267, Accuracy_test:
17.2 %
Epoch: 26, Loss_train: 2.203, Accuracy_train: 21.9 %, Loss_test 2.279, Accuracy_test:
16.8 %
Epoch: 27, Loss_train: 1.851, Accuracy_train: 37.5 %, Loss_test 2.234, Accuracy_test:
17.8 %
Epoch: 28, Loss_train: 1.911, Accuracy_train: 28.1 %, Loss_test 2.245, Accuracy_test:
18.3 %
Epoch: 29, Loss_train: 2.165, Accuracy_train: 28.1 %, Loss_test 2.241, Accuracy_test:
19.0 %
Epoch: 30, Loss_train: 2.081, Accuracy_train: 25.0 %, Loss_test 2.245, Accuracy_test:
16.6 %
Epoch: 31, Loss_train: 1.831, Accuracy_train: 31.2 %, Loss_test 2.224, Accuracy_test:
18.5 %
Epoch: 32, Loss_train: 1.817, Accuracy_train: 43.8 %, Loss_test 2.225, Accuracy_test:
18.8 %
Epoch: 33, Loss_train: 1.782, Accuracy_train: 40.6 %, Loss_test 2.219, Accuracy_test:
18.9 %
```

```
Epoch: 34, Loss_train: 2.055, Accuracy_train: 25.0 %, Loss_test 2.190, Accuracy_test:
19.5 %
Epoch: 35, Loss_train: 1.818, Accuracy_train: 43.8 %, Loss_test 2.173, Accuracy_test:
20.0 %
Epoch: 36, Loss_train: 1.991, Accuracy_train: 28.1 %, Loss_test 2.170, Accuracy_test:
18.9 %
Epoch: 37, Loss_train: 1.847, Accuracy_train: 21.9 %, Loss_test 2.194, Accuracy_test:
19.6 %
Epoch: 38, Loss_train: 1.937, Accuracy_train: 28.1 %, Loss_test 2.198, Accuracy_test:
19.3 %
Epoch: 39, Loss_train: 1.943, Accuracy_train: 18.8 %, Loss_test 2.152, Accuracy_test:
19.4 %
Epoch: 40, Loss_train: 1.918, Accuracy_train: 37.5 %, Loss_test 2.171, Accuracy_test:
20.4 %
Epoch: 41, Loss_train: 1.759, Accuracy_train: 37.5 %, Loss_test 2.152, Accuracy_test:
19.9 %
Epoch: 42, Loss_train: 1.621, Accuracy_train: 40.6 %, Loss_test 2.153, Accuracy_test:
20.2 %
Epoch: 43, Loss_train: 1.890, Accuracy_train: 28.1 %, Loss_test 2.153, Accuracy_test:
21.0 %
Epoch: 44, Loss_train: 1.828, Accuracy_train: 34.4 %, Loss_test 2.144, Accuracy_test:
20.6 %
Epoch: 45, Loss_train: 1.751, Accuracy_train: 37.5 %, Loss_test 2.143, Accuracy_test:
20.6 %
Epoch: 46, Loss_train: 1.774, Accuracy_train: 37.5 %, Loss_test 2.125, Accuracy_test:
22.1 %
Epoch: 47, Loss_train: 1.939, Accuracy_train: 43.8 %, Loss_test 2.139, Accuracy_test:
19.8 %
Epoch: 48, Loss_train: 2.001, Accuracy_train: 40.6 %, Loss_test 2.107, Accuracy_test:
21.2 %
Epoch: 49, Loss_train: 1.870, Accuracy_train: 28.1 %, Loss_test 2.084, Accuracy_test:
22.0 %
Epoch: 50, Loss_train: 1.579, Accuracy_train: 43.8 %, Loss_test 2.107, Accuracy_test:
22.1 %
```

Out[28]: <matplotlib.legend.Legend at 0x7fd6b420cb50>

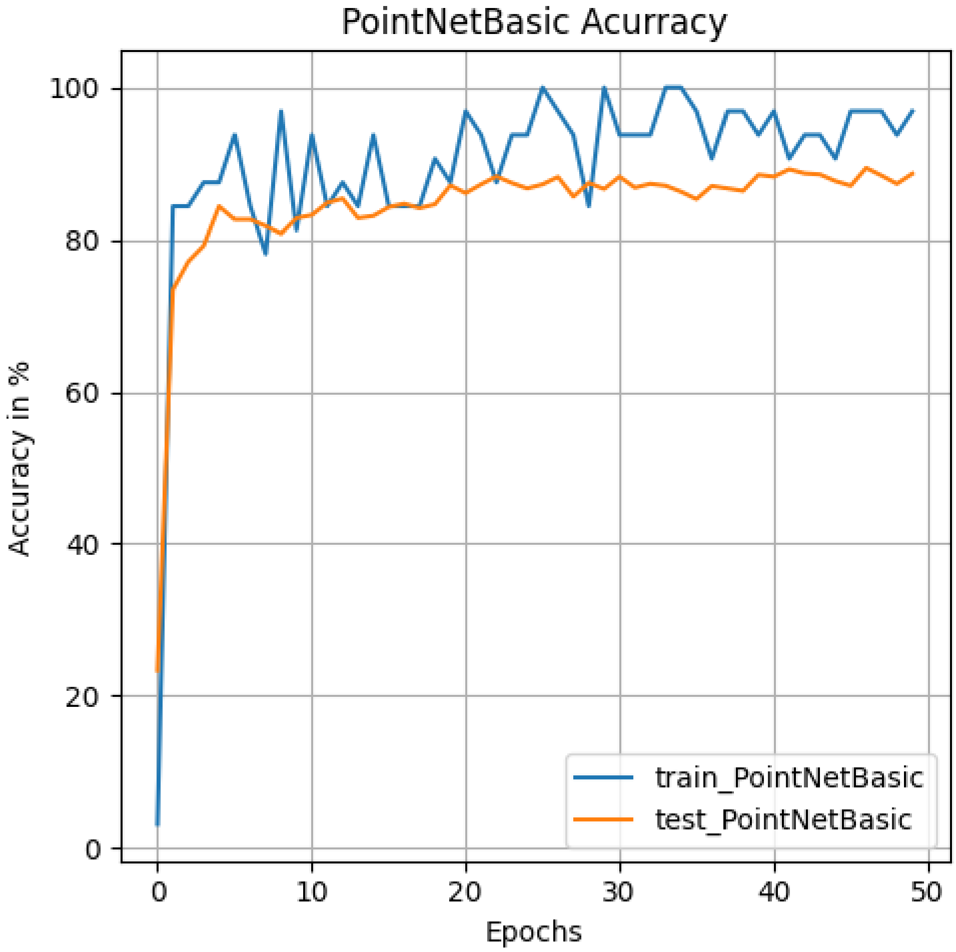


In [29]:
```
#PointBasic, no augment
model = PointNetBasic(NUM_POINTS,NUM_CLASSES)
(loss_train_array_POINTNetBasic,
 accuracy_train_array_POINTNetBasic,
 loss_test_array_POINTNetBasic,
 accuracy_test_array_POINTNetBasic) = train(
    model,
    dataloader_train,
    dataloader_test,
    epochs=50,
    loss_func=torch.nn.CrossEntropyLoss()
)




plt.figure(1)
plt.title("PointNetBasic Acurracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy in %")

plt.plot(accuracy_train_array_POINTNetBasic,label="train_PointNetBasic")
plt.plot(accuracy_test_array_POINTNetBasic,label="test_PointNetBasic")
plt.grid()
plt.legend()
```

```
Epoch: 1, Loss_train: 2.460, Accuracy_train: 3.1 %, Loss_test 2.155, Accuracy_test: 2
3.3 %
Epoch: 2, Loss_train: 0.481, Accuracy_train: 84.4 %, Loss_test 0.806, Accuracy_test:
73.5 %
Epoch: 3, Loss_train: 0.413, Accuracy_train: 84.4 %, Loss_test 0.660, Accuracy_test:
77.1 %
Epoch: 4, Loss_train: 0.316, Accuracy_train: 87.5 %, Loss_test 0.630, Accuracy_test:
79.2 %
Epoch: 5, Loss_train: 0.420, Accuracy_train: 87.5 %, Loss_test 0.493, Accuracy_test:
84.4 %
Epoch: 6, Loss_train: 0.244, Accuracy_train: 93.8 %, Loss_test 0.522, Accuracy_test:
82.7 %
Epoch: 7, Loss_train: 0.334, Accuracy_train: 84.4 %, Loss_test 0.501, Accuracy_test:
82.7 %
Epoch: 8, Loss_train: 0.494, Accuracy_train: 78.1 %, Loss_test 0.565, Accuracy_test:
81.8 %
Epoch: 9, Loss_train: 0.111, Accuracy_train: 96.9 %, Loss_test 0.600, Accuracy_test:
80.8 %
Epoch: 10, Loss_train: 0.733, Accuracy_train: 81.2 %, Loss_test 0.553, Accuracy_test:
82.8 %
Epoch: 11, Loss_train: 0.139, Accuracy_train: 93.8 %, Loss_test 0.498, Accuracy_test:
83.2 %
Epoch: 12, Loss_train: 0.485, Accuracy_train: 84.4 %, Loss_test 0.468, Accuracy_test:
84.8 %
Epoch: 13, Loss_train: 0.457, Accuracy_train: 87.5 %, Loss_test 0.427, Accuracy_test:
85.4 %
Epoch: 14, Loss_train: 0.387, Accuracy_train: 84.4 %, Loss_test 0.570, Accuracy_test:
82.8 %
Epoch: 15, Loss_train: 0.136, Accuracy_train: 93.8 %, Loss_test 0.523, Accuracy_test:
83.1 %
Epoch: 16, Loss_train: 0.275, Accuracy_train: 84.4 %, Loss_test 0.514, Accuracy_test:
84.3 %
Epoch: 17, Loss_train: 0.275, Accuracy_train: 84.4 %, Loss_test 0.468, Accuracy_test:
84.7 %
Epoch: 18, Loss_train: 0.355, Accuracy_train: 84.4 %, Loss_test 0.484, Accuracy_test:
84.1 %
Epoch: 19, Loss_train: 0.282, Accuracy_train: 90.6 %, Loss_test 0.491, Accuracy_test:
84.6 %
Epoch: 20, Loss_train: 0.315, Accuracy_train: 87.5 %, Loss_test 0.439, Accuracy_test:
87.1 %
Epoch: 21, Loss_train: 0.146, Accuracy_train: 96.9 %, Loss_test 0.439, Accuracy_test:
86.1 %
Epoch: 22, Loss_train: 0.140, Accuracy_train: 93.8 %, Loss_test 0.425, Accuracy_test:
87.2 %
Epoch: 23, Loss_train: 0.312, Accuracy_train: 87.5 %, Loss_test 0.409, Accuracy_test:
88.3 %
Epoch: 24, Loss_train: 0.225, Accuracy_train: 93.8 %, Loss_test 0.425, Accuracy_test:
87.4 %
Epoch: 25, Loss_train: 0.139, Accuracy_train: 93.8 %, Loss_test 0.408, Accuracy_test:
86.7 %
Epoch: 26, Loss_train: 0.065, Accuracy_train: 100.0 %, Loss_test 0.410, Accuracy_tes
t: 87.2 %
Epoch: 27, Loss_train: 0.091, Accuracy_train: 96.9 %, Loss_test 0.438, Accuracy_test:
88.2 %
Epoch: 28, Loss_train: 0.210, Accuracy_train: 93.8 %, Loss_test 0.477, Accuracy_test:
85.6 %
Epoch: 29, Loss_train: 0.413, Accuracy_train: 84.4 %, Loss_test 0.422, Accuracy_test:
87.4 %
Epoch: 30, Loss_train: 0.042, Accuracy_train: 100.0 %, Loss_test 0.437, Accuracy_tes
t: 86.6 %
Epoch: 31, Loss_train: 0.142, Accuracy_train: 93.8 %, Loss_test 0.425, Accuracy_test:
88.3 %
Epoch: 32, Loss_train: 0.089, Accuracy_train: 93.8 %, Loss_test 0.392, Accuracy_test:
86.8 %
Epoch: 33, Loss_train: 0.113, Accuracy_train: 93.8 %, Loss_test 0.451, Accuracy_test:
87.3 %
```

```
Epoch: 34, Loss_train: 0.042, Accuracy_train: 100.0 %, Loss_test 0.445, Accuracy_tes
t: 87.1 %
Epoch: 35, Loss_train: 0.042, Accuracy_train: 100.0 %, Loss_test 0.446, Accuracy_tes
t: 86.3 %
Epoch: 36, Loss_train: 0.058, Accuracy_train: 96.9 %, Loss_test 0.546, Accuracy_test:
85.3 %
Epoch: 37, Loss_train: 0.147, Accuracy_train: 90.6 %, Loss_test 0.431, Accuracy_test:
87.0 %
Epoch: 38, Loss_train: 0.085, Accuracy_train: 96.9 %, Loss_test 0.469, Accuracy_test:
86.7 %
Epoch: 39, Loss_train: 0.102, Accuracy_train: 96.9 %, Loss_test 0.466, Accuracy_test:
86.4 %
Epoch: 40, Loss_train: 0.203, Accuracy_train: 93.8 %, Loss_test 0.401, Accuracy_test:
88.5 %
Epoch: 41, Loss_train: 0.094, Accuracy_train: 96.9 %, Loss_test 0.435, Accuracy_test:
88.3 %
Epoch: 42, Loss_train: 0.282, Accuracy_train: 90.6 %, Loss_test 0.388, Accuracy_test:
89.2 %
Epoch: 43, Loss_train: 0.261, Accuracy_train: 93.8 %, Loss_test 0.408, Accuracy_test:
88.7 %
Epoch: 44, Loss_train: 0.106, Accuracy_train: 93.8 %, Loss_test 0.398, Accuracy_test:
88.5 %
Epoch: 45, Loss_train: 0.208, Accuracy_train: 90.6 %, Loss_test 0.432, Accuracy_test:
87.7 %
Epoch: 46, Loss_train: 0.053, Accuracy_train: 96.9 %, Loss_test 0.443, Accuracy_test:
87.1 %
Epoch: 47, Loss_train: 0.055, Accuracy_train: 96.9 %, Loss_test 0.426, Accuracy_test:
89.4 %
Epoch: 48, Loss_train: 0.134, Accuracy_train: 96.9 %, Loss_test 0.402, Accuracy_test:
88.4 %
Epoch: 49, Loss_train: 0.082, Accuracy_train: 93.8 %, Loss_test 0.475, Accuracy_test:
87.3 %
Epoch: 50, Loss_train: 0.088, Accuracy_train: 96.9 %, Loss_test 0.423, Accuracy_test:
88.6 %
```

```
Out[29]: <matplotlib.legend.Legend at 0x7fd6b41a1670>
```

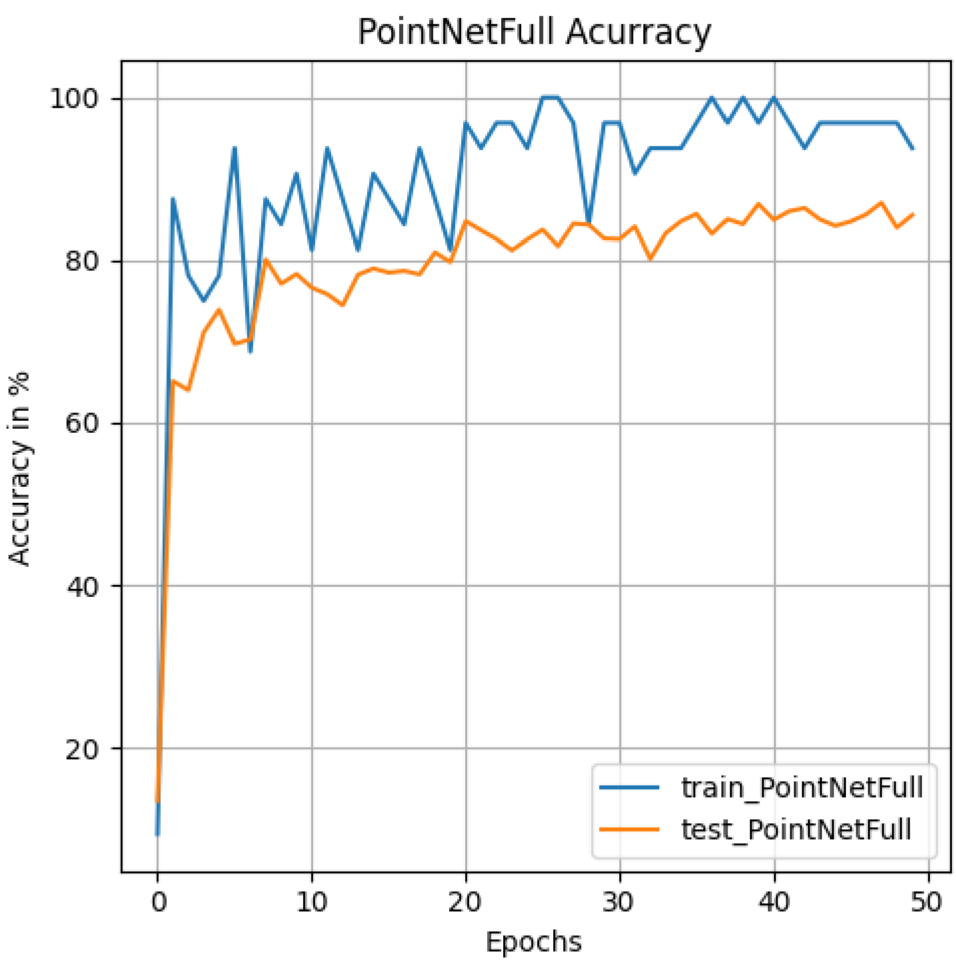


In [30]:
```
#PointFull, no augment
model = PointNetFull(NUM_POINTS,NUM_CLASSES)
(loss_train_array_POINTNetFull,
 accuracy_train_array_POINTNetFull,
 loss_test_array_POINTNetFull,
 accuracy_test_array_POINTNetFull) = train(
    model,
    dataloader_train,
    dataloader_test,
    epochs=50,
    loss_func=torch.nn.CrossEntropyLoss()
)

plt.figure(1)
plt.title("PointNetFull Acurracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy in %")

plt.plot(accuracy_train_array_POINTNetFull,label="train_PointNetFull")
plt.plot(accuracy_test_array_POINTNetFull,label="test_PointNetFull")
plt.grid()
plt.legend()
```

```
Epoch: 1, Loss_train: 2.399, Accuracy_train: 9.4 %, Loss_test 2.378, Accuracy_test: 13.5 %
Epoch: 2, Loss_train: 0.525, Accuracy_train: 87.5 %, Loss_test 1.142, Accuracy_test: 65.1 %
Epoch: 3, Loss_train: 0.637, Accuracy_train: 78.1 %, Loss_test 1.090, Accuracy_test: 64.0 %
Epoch: 4, Loss_train: 0.682, Accuracy_train: 75.0 %, Loss_test 0.842, Accuracy_test: 71.1 %
Epoch: 5, Loss_train: 0.576, Accuracy_train: 78.1 %, Loss_test 0.817, Accuracy_test: 73.9 %
Epoch: 6, Loss_train: 0.356, Accuracy_train: 93.8 %, Loss_test 0.966, Accuracy_test: 69.7 %
Epoch: 7, Loss_train: 0.864, Accuracy_train: 68.8 %, Loss_test 0.889, Accuracy_test: 70.3 %
Epoch: 8, Loss_train: 0.438, Accuracy_train: 87.5 %, Loss_test 0.688, Accuracy_test: 80.1 %
Epoch: 9, Loss_train: 0.411, Accuracy_train: 84.4 %, Loss_test 0.767, Accuracy_test: 77.1 %
Epoch: 10, Loss_train: 0.174, Accuracy_train: 90.6 %, Loss_test 0.722, Accuracy_test: 78.3 %
Epoch: 11, Loss_train: 0.666, Accuracy_train: 81.2 %, Loss_test 0.784, Accuracy_test: 76.6 %
Epoch: 12, Loss_train: 0.153, Accuracy_train: 93.8 %, Loss_test 0.783, Accuracy_test: 75.8 %
Epoch: 13, Loss_train: 0.411, Accuracy_train: 87.5 %, Loss_test 0.846, Accuracy_test: 74.5 %
Epoch: 14, Loss_train: 0.402, Accuracy_train: 81.2 %, Loss_test 0.736, Accuracy_test: 78.2 %
Epoch: 15, Loss_train: 0.312, Accuracy_train: 90.6 %, Loss_test 0.663, Accuracy_test: 79.0 %
Epoch: 16, Loss_train: 0.297, Accuracy_train: 87.5 %, Loss_test 0.673, Accuracy_test: 78.5 %
Epoch: 17, Loss_train: 0.432, Accuracy_train: 84.4 %, Loss_test 0.730, Accuracy_test: 78.7 %
Epoch: 18, Loss_train: 0.146, Accuracy_train: 93.8 %, Loss_test 0.718, Accuracy_test: 78.3 %
Epoch: 19, Loss_train: 0.365, Accuracy_train: 87.5 %, Loss_test 0.567, Accuracy_test: 81.0 %
Epoch: 20, Loss_train: 0.639, Accuracy_train: 81.2 %, Loss_test 0.657, Accuracy_test: 79.8 %
Epoch: 21, Loss_train: 0.071, Accuracy_train: 96.9 %, Loss_test 0.540, Accuracy_test: 84.8 %
Epoch: 22, Loss_train: 0.208, Accuracy_train: 93.8 %, Loss_test 0.561, Accuracy_test: 83.7 %
Epoch: 23, Loss_train: 0.098, Accuracy_train: 96.9 %, Loss_test 0.540, Accuracy_test: 82.6 %
Epoch: 24, Loss_train: 0.111, Accuracy_train: 96.9 %, Loss_test 0.604, Accuracy_test: 81.2 %
Epoch: 25, Loss_train: 0.144, Accuracy_train: 93.8 %, Loss_test 0.577, Accuracy_test: 82.5 %
Epoch: 26, Loss_train: 0.040, Accuracy_train: 100.0 %, Loss_test 0.546, Accuracy_test: 83.7 %
Epoch: 27, Loss_train: 0.067, Accuracy_train: 100.0 %, Loss_test 0.598, Accuracy_test: 81.7 %
Epoch: 28, Loss_train: 0.321, Accuracy_train: 96.9 %, Loss_test 0.489, Accuracy_test: 84.5 %
Epoch: 29, Loss_train: 0.381, Accuracy_train: 84.4 %, Loss_test 0.519, Accuracy_test: 84.3 %
Epoch: 30, Loss_train: 0.193, Accuracy_train: 96.9 %, Loss_test 0.539, Accuracy_test: 82.7 %
Epoch: 31, Loss_train: 0.150, Accuracy_train: 96.9 %, Loss_test 0.581, Accuracy_test: 82.6 %
Epoch: 32, Loss_train: 0.191, Accuracy_train: 90.6 %, Loss_test 0.551, Accuracy_test: 84.1 %
Epoch: 33, Loss_train: 0.198, Accuracy_train: 93.8 %, Loss_test 0.613, Accuracy_test: 80.2 %
```

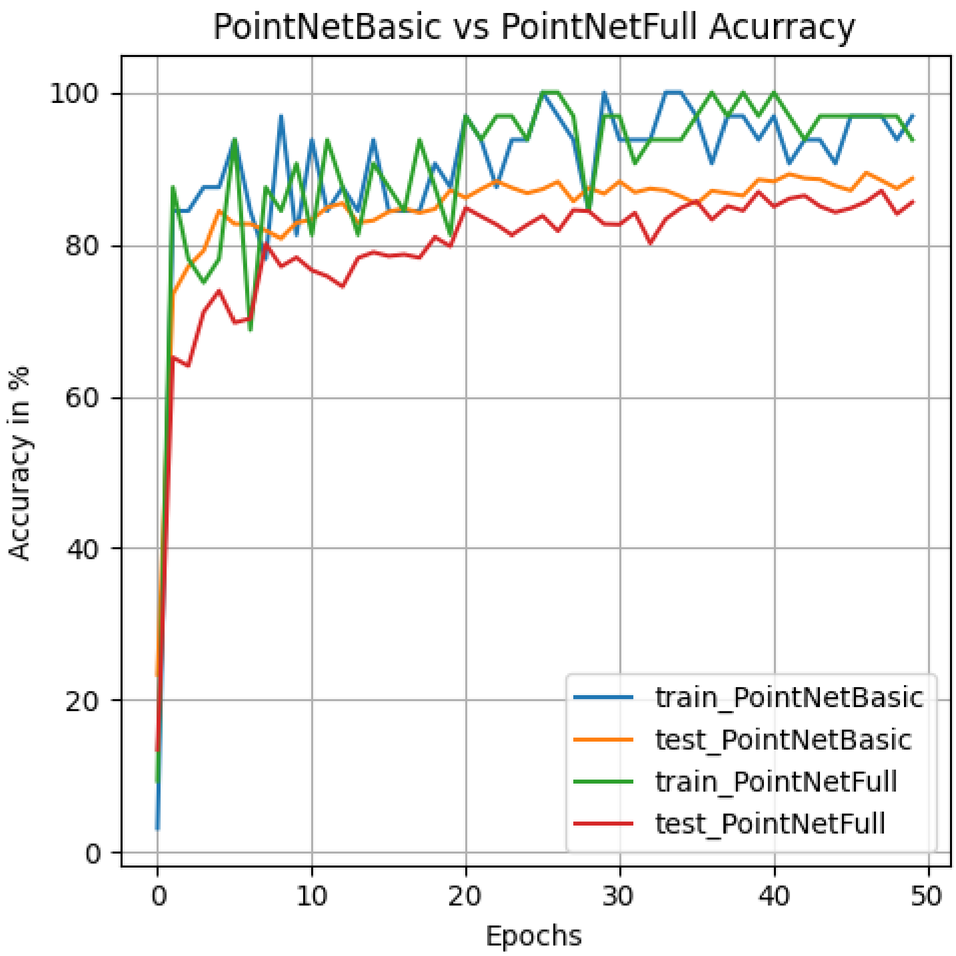
```
Epoch: 34, Loss_train: 0.326, Accuracy_train: 93.8 %, Loss_test 0.534, Accuracy_test: 83.3 %
Epoch: 35, Loss_train: 0.132, Accuracy_train: 93.8 %, Loss_test 0.489, Accuracy_test: 84.8 %
Epoch: 36, Loss_train: 0.143, Accuracy_train: 96.9 %, Loss_test 0.471, Accuracy_test: 85.7 %
Epoch: 37, Loss_train: 0.076, Accuracy_train: 100.0 %, Loss_test 0.572, Accuracy_test: 83.2 %
Epoch: 38, Loss_train: 0.118, Accuracy_train: 96.9 %, Loss_test 0.549, Accuracy_test: 85.0 %
Epoch: 39, Loss_train: 0.044, Accuracy_train: 100.0 %, Loss_test 0.574, Accuracy_test: 84.4 %
Epoch: 40, Loss_train: 0.090, Accuracy_train: 96.9 %, Loss_test 0.456, Accuracy_test: 86.9 %
Epoch: 41, Loss_train: 0.031, Accuracy_train: 100.0 %, Loss_test 0.515, Accuracy_test: 84.9 %
Epoch: 42, Loss_train: 0.109, Accuracy_train: 96.9 %, Loss_test 0.467, Accuracy_test: 86.0 %
Epoch: 43, Loss_train: 0.194, Accuracy_train: 93.8 %, Loss_test 0.468, Accuracy_test: 86.4 %
Epoch: 44, Loss_train: 0.083, Accuracy_train: 96.9 %, Loss_test 0.484, Accuracy_test: 85.0 %
Epoch: 45, Loss_train: 0.097, Accuracy_train: 96.9 %, Loss_test 0.550, Accuracy_test: 84.2 %
Epoch: 46, Loss_train: 0.044, Accuracy_train: 96.9 %, Loss_test 0.568, Accuracy_test: 84.7 %
Epoch: 47, Loss_train: 0.049, Accuracy_train: 96.9 %, Loss_test 0.514, Accuracy_test: 85.6 %
Epoch: 48, Loss_train: 0.103, Accuracy_train: 96.9 %, Loss_test 0.439, Accuracy_test: 87.0 %
Epoch: 49, Loss_train: 0.072, Accuracy_train: 96.9 %, Loss_test 0.547, Accuracy_test: 84.0 %
Epoch: 50, Loss_train: 0.091, Accuracy_train: 93.8 %, Loss_test 0.514, Accuracy_test: 85.5 %
```

Out[30]: <matplotlib.legend.Legend at 0x7fd6b411e5e0>



In [31]:
```
plt.figure(1)
plt.title("PointNetBasic vs PointNetFull Acurracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy in %")
plt.plot(accuracy_train_array_POINTNetBasic,label="train_PointNetBasic")
plt.plot(accuracy_test_array_POINTNetBasic,label="test_PointNetBasic")
plt.plot(accuracy_train_array_POINTNetFull,label="train_PointNetFull")
plt.plot(accuracy_test_array_POINTNetFull,label="test_PointNetFull")
plt.grid()
plt.legend()
```

Out[31]: <matplotlib.legend.Legend at 0x7fd6b40e2280>



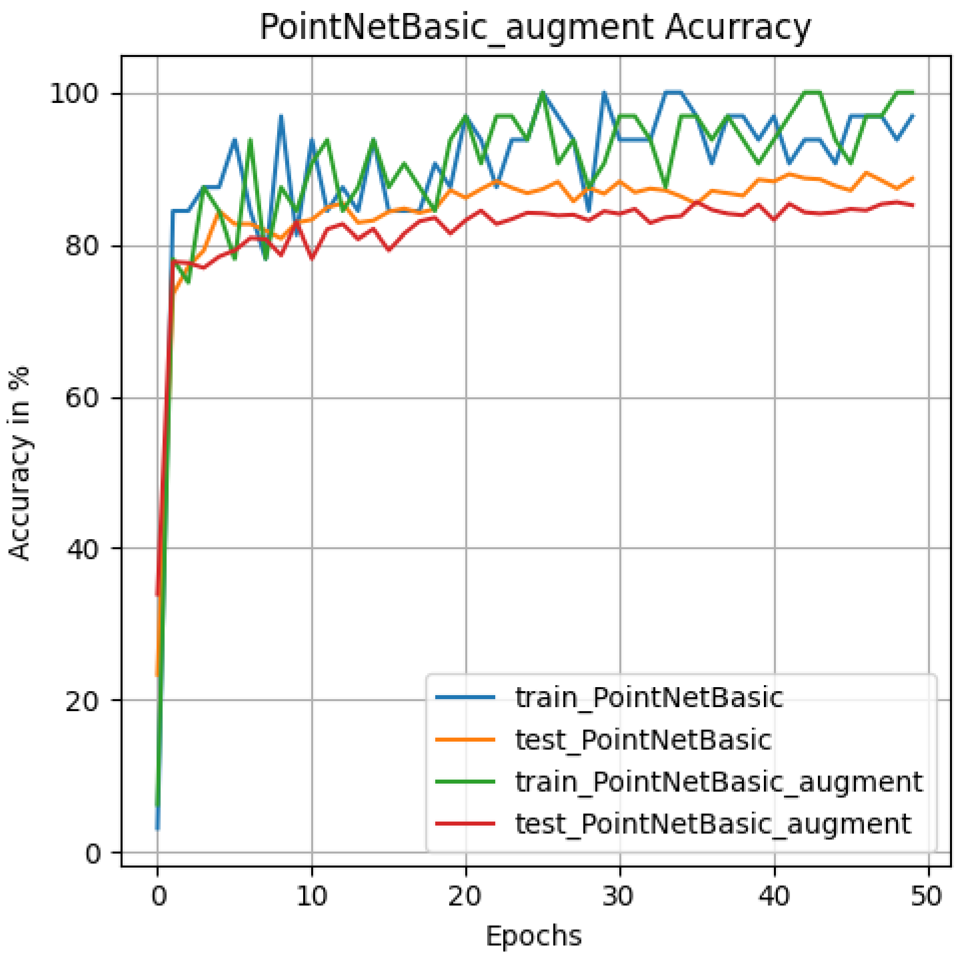
In [32]:
```
#PointFull, augment, AxisReducing
model = PointNetBasic(NUM_POINTS,NUM_CLASSES)
(loss_train_array_POINTNetBasic_augment,
 accuracy_train_array_POINTNetBasic_augment,
 loss_test_array_POINTNetBasic_augment,
 accuracy_test_array_POINTNetBasic_augment) = train(
    model,
    dataloader_train_augment,
    dataloader_test_augment,
    epochs=50,
    loss_func=torch.nn.CrossEntropyLoss()
)

plt.figure(1)
plt.title("PointNetBasic_augment Acurracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy in %")

plt.plot(accuracy_train_array_POINTNetBasic,label="train_PointNetBasic")
plt.plot(accuracy_test_array_POINTNetBasic,label="test_PointNetBasic")

plt.plot(accuracy_train_array_POINTNetBasic_augment,label="train_PointNetBasic_augmen
plt.plot(accuracy_test_array_POINTNetBasic_augment,label="test_PointNetBasic_augment"
```

```
plt.grid()
plt.legend()
```

```
Epoch: 1, Loss_train: 2.510, Accuracy_train: 6.2 %, Loss_test 1.984, Accuracy_test: 3
3.9 %
Epoch: 2, Loss_train: 0.629, Accuracy_train: 78.1 %, Loss_test 0.696, Accuracy_test:
77.7 %
Epoch: 3, Loss_train: 0.518, Accuracy_train: 75.0 %, Loss_test 0.685, Accuracy_test:
77.6 %
Epoch: 4, Loss_train: 0.373, Accuracy_train: 87.5 %, Loss_test 0.677, Accuracy_test:
76.9 %
Epoch: 5, Loss_train: 0.504, Accuracy_train: 84.4 %, Loss_test 0.639, Accuracy_test:
78.4 %
Epoch: 6, Loss_train: 0.538, Accuracy_train: 78.1 %, Loss_test 0.611, Accuracy_test:
79.2 %
Epoch: 7, Loss_train: 0.114, Accuracy_train: 93.8 %, Loss_test 0.619, Accuracy_test:
80.9 %
Epoch: 8, Loss_train: 0.413, Accuracy_train: 78.1 %, Loss_test 0.640, Accuracy_test:
80.7 %
Epoch: 9, Loss_train: 0.284, Accuracy_train: 87.5 %, Loss_test 0.672, Accuracy_test:
78.6 %
Epoch: 10, Loss_train: 0.450, Accuracy_train: 84.4 %, Loss_test 0.557, Accuracy_test:
82.9 %
Epoch: 11, Loss_train: 0.208, Accuracy_train: 90.6 %, Loss_test 0.659, Accuracy_test:
78.1 %
Epoch: 12, Loss_train: 0.237, Accuracy_train: 93.8 %, Loss_test 0.612, Accuracy_test:
82.0 %
Epoch: 13, Loss_train: 0.283, Accuracy_train: 84.4 %, Loss_test 0.511, Accuracy_test:
82.7 %
Epoch: 14, Loss_train: 0.210, Accuracy_train: 87.5 %, Loss_test 0.613, Accuracy_test:
80.7 %
Epoch: 15, Loss_train: 0.128, Accuracy_train: 93.8 %, Loss_test 0.626, Accuracy_test:
82.0 %
Epoch: 16, Loss_train: 0.253, Accuracy_train: 87.5 %, Loss_test 0.649, Accuracy_test:
79.2 %
Epoch: 17, Loss_train: 0.364, Accuracy_train: 90.6 %, Loss_test 0.640, Accuracy_test:
81.4 %
Epoch: 18, Loss_train: 0.249, Accuracy_train: 87.5 %, Loss_test 0.553, Accuracy_test:
83.0 %
Epoch: 19, Loss_train: 0.322, Accuracy_train: 84.4 %, Loss_test 0.571, Accuracy_test:
83.5 %
Epoch: 20, Loss_train: 0.132, Accuracy_train: 93.8 %, Loss_test 0.647, Accuracy_test:
81.4 %
Epoch: 21, Loss_train: 0.141, Accuracy_train: 96.9 %, Loss_test 0.565, Accuracy_test:
83.2 %
Epoch: 22, Loss_train: 0.206, Accuracy_train: 90.6 %, Loss_test 0.528, Accuracy_test:
84.4 %
Epoch: 23, Loss_train: 0.074, Accuracy_train: 96.9 %, Loss_test 0.611, Accuracy_test:
82.7 %
Epoch: 24, Loss_train: 0.044, Accuracy_train: 96.9 %, Loss_test 0.579, Accuracy_test:
83.3 %
Epoch: 25, Loss_train: 0.190, Accuracy_train: 93.8 %, Loss_test 0.574, Accuracy_test:
84.1 %
Epoch: 26, Loss_train: 0.040, Accuracy_train: 100.0 %, Loss_test 0.565, Accuracy_tes
t: 84.1 %
Epoch: 27, Loss_train: 0.166, Accuracy_train: 90.6 %, Loss_test 0.590, Accuracy_test:
83.8 %
Epoch: 28, Loss_train: 0.110, Accuracy_train: 93.8 %, Loss_test 0.615, Accuracy_test:
83.9 %
Epoch: 29, Loss_train: 0.234, Accuracy_train: 87.5 %, Loss_test 0.602, Accuracy_test:
83.1 %
Epoch: 30, Loss_train: 0.167, Accuracy_train: 90.6 %, Loss_test 0.593, Accuracy_test:
84.3 %
Epoch: 31, Loss_train: 0.103, Accuracy_train: 96.9 %, Loss_test 0.617, Accuracy_test:
84.0 %
Epoch: 32, Loss_train: 0.061, Accuracy_train: 96.9 %, Loss_test 0.571, Accuracy_test:
84.6 %
Epoch: 33, Loss_train: 0.113, Accuracy_train: 93.8 %, Loss_test 0.655, Accuracy_test:
82.8 %
```

```
Epoch: 34, Loss_train: 0.179, Accuracy_train: 87.5 %, Loss_test 0.610, Accuracy_test:
83.5 %
Epoch: 35, Loss_train: 0.064, Accuracy_train: 96.9 %, Loss_test 0.619, Accuracy_test:
83.7 %
Epoch: 36, Loss_train: 0.083, Accuracy_train: 96.9 %, Loss_test 0.538, Accuracy_test:
85.6 %
Epoch: 37, Loss_train: 0.102, Accuracy_train: 93.8 %, Loss_test 0.590, Accuracy_test:
84.5 %
Epoch: 38, Loss_train: 0.054, Accuracy_train: 96.9 %, Loss_test 0.589, Accuracy_test:
84.0 %
Epoch: 39, Loss_train: 0.164, Accuracy_train: 93.8 %, Loss_test 0.655, Accuracy_test:
83.8 %
Epoch: 40, Loss_train: 0.107, Accuracy_train: 90.6 %, Loss_test 0.582, Accuracy_test:
85.2 %
Epoch: 41, Loss_train: 0.139, Accuracy_train: 93.8 %, Loss_test 0.651, Accuracy_test:
83.2 %
Epoch: 42, Loss_train: 0.074, Accuracy_train: 96.9 %, Loss_test 0.576, Accuracy_test:
85.3 %
Epoch: 43, Loss_train: 0.029, Accuracy_train: 100.0 %, Loss_test 0.670, Accuracy_tes
t: 84.2 %
Epoch: 44, Loss_train: 0.039, Accuracy_train: 100.0 %, Loss_test 0.650, Accuracy_tes
t: 84.0 %
Epoch: 45, Loss_train: 0.119, Accuracy_train: 93.8 %, Loss_test 0.645, Accuracy_test:
84.2 %
Epoch: 46, Loss_train: 0.175, Accuracy_train: 90.6 %, Loss_test 0.613, Accuracy_test:
84.6 %
Epoch: 47, Loss_train: 0.080, Accuracy_train: 96.9 %, Loss_test 0.618, Accuracy_test:
84.4 %
Epoch: 48, Loss_train: 0.077, Accuracy_train: 96.9 %, Loss_test 0.646, Accuracy_test:
85.3 %
Epoch: 49, Loss_train: 0.006, Accuracy_train: 100.0 %, Loss_test 0.618, Accuracy_tes
t: 85.5 %
Epoch: 50, Loss_train: 0.024, Accuracy_train: 100.0 %, Loss_test 0.692, Accuracy_tes
t: 85.1 %
```

Out[32]: <matplotlib.legend.Legend at 0x7fd6b077fee0>

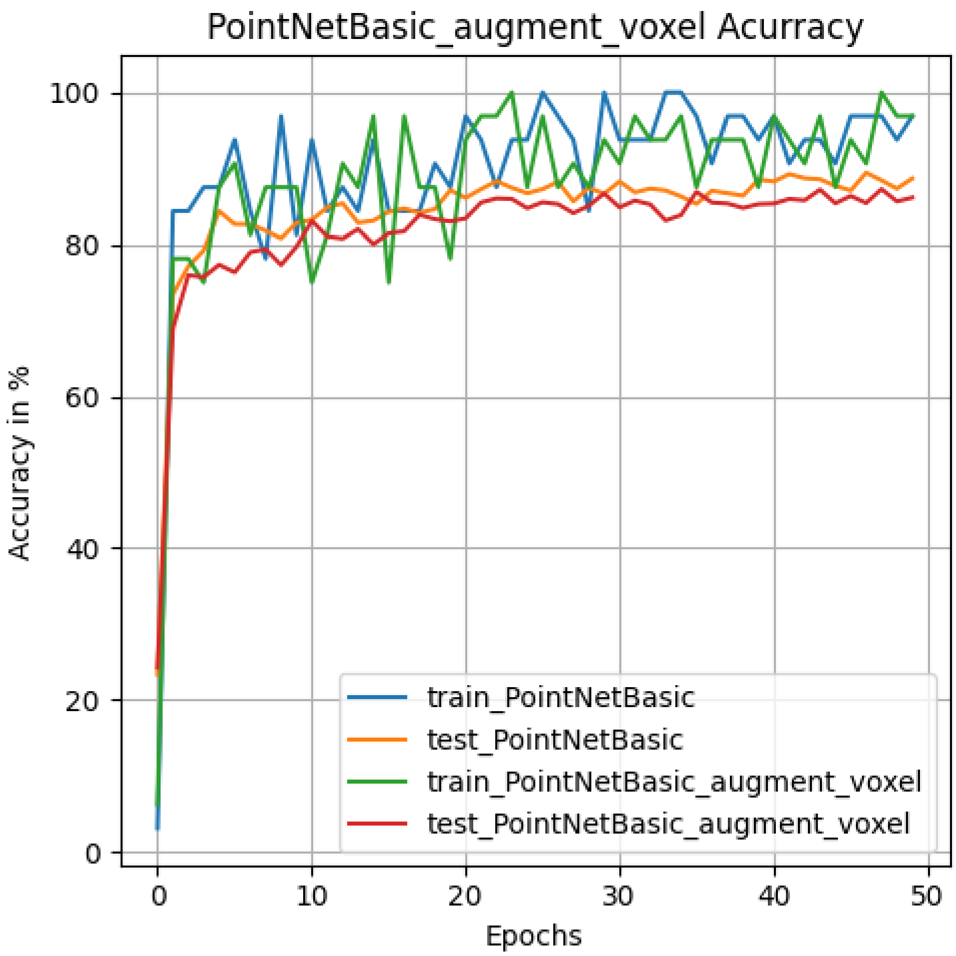


In [33]:
```
#PointNetBasic, augment, voxel
model = PointNetBasic(NUM_POINTS,NUM_CLASSES)
(loss_train_array_POINTNetBasic_augment_voxel,
 accuracy_train_array_POINTNetBasic_augment_voxel,
 loss_test_array_POINTNetBasic_augment_voxel,
 accuracy_test_array_POINTNetBasic_augment_voxel) = train(
    model,
    dataloader_train_augment_voxel,
    dataloader_test_augment_voxel,
    epochs=50,
    loss_func=torch.nn.CrossEntropyLoss()
)

plt.figure(1)
plt.title("PointNetBasic_augment_voxel Acurracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy in %")

plt.plot(accuracy_train_array_POINTNetBasic,label="train_PointNetBasic")
plt.plot(accuracy_test_array_POINTNetBasic,label="test_PointNetBasic")

plt.plot(accuracy_train_array_POINTNetBasic_augment_voxel,label="train_PointNetBasic_
plt.plot(accuracy_test_array_POINTNetBasic_augment_voxel,label="test_PointNetBasic_au
plt.grid()
plt.legend()
```

```
Epoch: 1, Loss_train: 2.418, Accuracy_train: 6.2 %, Loss_test 2.123, Accuracy_test: 2
4.2 %
Epoch: 2, Loss_train: 0.680, Accuracy_train: 78.1 %, Loss_test 0.892, Accuracy_test:
68.9 %
Epoch: 3, Loss_train: 0.577, Accuracy_train: 78.1 %, Loss_test 0.784, Accuracy_test:
76.0 %
Epoch: 4, Loss_train: 0.629, Accuracy_train: 75.0 %, Loss_test 0.709, Accuracy_test:
75.7 %
Epoch: 5, Loss_train: 0.425, Accuracy_train: 87.5 %, Loss_test 0.650, Accuracy_test:
77.3 %
Epoch: 6, Loss_train: 0.405, Accuracy_train: 90.6 %, Loss_test 0.695, Accuracy_test:
76.4 %
Epoch: 7, Loss_train: 0.434, Accuracy_train: 81.2 %, Loss_test 0.634, Accuracy_test:
79.0 %
Epoch: 8, Loss_train: 0.333, Accuracy_train: 87.5 %, Loss_test 0.588, Accuracy_test:
79.4 %
Epoch: 9, Loss_train: 0.372, Accuracy_train: 87.5 %, Loss_test 0.677, Accuracy_test:
77.3 %
Epoch: 10, Loss_train: 0.400, Accuracy_train: 87.5 %, Loss_test 0.630, Accuracy_test:
79.7 %
Epoch: 11, Loss_train: 0.722, Accuracy_train: 75.0 %, Loss_test 0.541, Accuracy_test:
83.1 %
Epoch: 12, Loss_train: 0.493, Accuracy_train: 81.2 %, Loss_test 0.568, Accuracy_test:
81.0 %
Epoch: 13, Loss_train: 0.373, Accuracy_train: 90.6 %, Loss_test 0.598, Accuracy_test:
80.7 %
Epoch: 14, Loss_train: 0.491, Accuracy_train: 87.5 %, Loss_test 0.540, Accuracy_test:
82.0 %
Epoch: 15, Loss_train: 0.146, Accuracy_train: 96.9 %, Loss_test 0.617, Accuracy_test:
80.0 %
Epoch: 16, Loss_train: 0.673, Accuracy_train: 75.0 %, Loss_test 0.573, Accuracy_test:
81.5 %
Epoch: 17, Loss_train: 0.140, Accuracy_train: 96.9 %, Loss_test 0.592, Accuracy_test:
81.7 %
Epoch: 18, Loss_train: 0.337, Accuracy_train: 87.5 %, Loss_test 0.527, Accuracy_test:
83.9 %
Epoch: 19, Loss_train: 0.480, Accuracy_train: 87.5 %, Loss_test 0.584, Accuracy_test:
83.3 %
Epoch: 20, Loss_train: 0.389, Accuracy_train: 78.1 %, Loss_test 0.544, Accuracy_test:
83.0 %
Epoch: 21, Loss_train: 0.221, Accuracy_train: 93.8 %, Loss_test 0.521, Accuracy_test:
83.4 %
Epoch: 22, Loss_train: 0.161, Accuracy_train: 96.9 %, Loss_test 0.485, Accuracy_test:
85.5 %
Epoch: 23, Loss_train: 0.149, Accuracy_train: 96.9 %, Loss_test 0.463, Accuracy_test:
86.0 %
Epoch: 24, Loss_train: 0.070, Accuracy_train: 100.0 %, Loss_test 0.489, Accuracy_tes
t: 86.0 %
Epoch: 25, Loss_train: 0.200, Accuracy_train: 87.5 %, Loss_test 0.472, Accuracy_test:
84.7 %
Epoch: 26, Loss_train: 0.232, Accuracy_train: 96.9 %, Loss_test 0.467, Accuracy_test:
85.5 %
Epoch: 27, Loss_train: 0.292, Accuracy_train: 87.5 %, Loss_test 0.494, Accuracy_test:
85.3 %
Epoch: 28, Loss_train: 0.172, Accuracy_train: 90.6 %, Loss_test 0.497, Accuracy_test:
84.1 %
Epoch: 29, Loss_train: 0.258, Accuracy_train: 87.5 %, Loss_test 0.503, Accuracy_test:
85.1 %
Epoch: 30, Loss_train: 0.107, Accuracy_train: 93.8 %, Loss_test 0.434, Accuracy_test:
86.7 %
Epoch: 31, Loss_train: 0.299, Accuracy_train: 90.6 %, Loss_test 0.484, Accuracy_test:
84.8 %
Epoch: 32, Loss_train: 0.073, Accuracy_train: 96.9 %, Loss_test 0.467, Accuracy_test:
85.7 %
Epoch: 33, Loss_train: 0.321, Accuracy_train: 93.8 %, Loss_test 0.497, Accuracy_test:
85.2 %
```

```
Epoch: 34, Loss_train: 0.167, Accuracy_train: 93.8 %, Loss_test 0.528, Accuracy_test: 
83.1 %
Epoch: 35, Loss_train: 0.071, Accuracy_train: 96.9 %, Loss_test 0.541, Accuracy_test: 
83.8 %
Epoch: 36, Loss_train: 0.257, Accuracy_train: 87.5 %, Loss_test 0.478, Accuracy_test: 
86.9 %
Epoch: 37, Loss_train: 0.208, Accuracy_train: 93.8 %, Loss_test 0.528, Accuracy_test: 
85.5 %
Epoch: 38, Loss_train: 0.184, Accuracy_train: 93.8 %, Loss_test 0.496, Accuracy_test: 
85.3 %
Epoch: 39, Loss_train: 0.268, Accuracy_train: 93.8 %, Loss_test 0.454, Accuracy_test: 
84.8 %
Epoch: 40, Loss_train: 0.244, Accuracy_train: 87.5 %, Loss_test 0.445, Accuracy_test: 
85.3 %
Epoch: 41, Loss_train: 0.086, Accuracy_train: 96.9 %, Loss_test 0.465, Accuracy_test: 
85.3 %
Epoch: 42, Loss_train: 0.116, Accuracy_train: 93.8 %, Loss_test 0.442, Accuracy_test: 
86.0 %
Epoch: 43, Loss_train: 0.243, Accuracy_train: 90.6 %, Loss_test 0.471, Accuracy_test: 
85.8 %
Epoch: 44, Loss_train: 0.133, Accuracy_train: 96.9 %, Loss_test 0.473, Accuracy_test: 
87.2 %
Epoch: 45, Loss_train: 0.304, Accuracy_train: 87.5 %, Loss_test 0.475, Accuracy_test: 
85.4 %
Epoch: 46, Loss_train: 0.150, Accuracy_train: 93.8 %, Loss_test 0.467, Accuracy_test: 
86.4 %
Epoch: 47, Loss_train: 0.173, Accuracy_train: 90.6 %, Loss_test 0.509, Accuracy_test: 
85.4 %
Epoch: 48, Loss_train: 0.062, Accuracy_train: 100.0 %, Loss_test 0.406, Accuracy_tes
t: 87.2 %
Epoch: 49, Loss_train: 0.313, Accuracy_train: 96.9 %, Loss_test 0.444, Accuracy_test: 
85.6 %
Epoch: 50, Loss_train: 0.183, Accuracy_train: 96.9 %, Loss_test 0.474, Accuracy_test: 
86.1 %
```

Out[33]: `<matplotlib.legend.Legend at 0x7fd6b45a70a0>`



The model :

In [121…

```
model = PointMLP(NUM_POINTS,NUM_CLASSES)
print(model)
```

```
PointMLP(
  (fc_1): Linear(in_features=3072, out_features=512, bias=True)
  (bn_1): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
  (fc_2): Linear(in_features=512, out_features=256, bias=True)
  (dropout_1): Dropout(p=0.3, inplace=False)
  (bn_2): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
  (fc_3): Linear(in_features=256, out_features=1024, bias=True)
  (bn_3): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats
=True)
)
```

In [122…

```
model = PointNetBasic(NUM_POINTS,NUM_CLASSES)
print(model)
```

```
PointNetBasic(
  (fc_1): Conv1d(3, 32, kernel_size=(1,), stride=(1,))
  (bn_1): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
  (fc_2): Conv1d(32, 32, kernel_size=(1,), stride=(1,))
  (bn_2): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
  (fc_3): Conv1d(32, 32, kernel_size=(1,), stride=(1,))
  (bn_3): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
  (fc_4): Conv1d(32, 128, kernel_size=(1,), stride=(1,))
  (bn_4): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
  (fc_5): Conv1d(128, 512, kernel_size=(1,), stride=(1,))
  (bn_5): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
  (mp): MaxPool1d(kernel_size=512, stride=512, padding=0, dilation=1, ceil_mode=Fals
e)
  (fc_6): Linear(in_features=1024, out_features=256, bias=True)
  (bn_6): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
  (fc_7): Linear(in_features=256, out_features=128, bias=True)
  (dropout_1): Dropout(p=0.3, inplace=False)
  (bn_7): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
  (fc_8): Linear(in_features=128, out_features=10, bias=True)
)
```

In [123…

```
model = PointNetFull(NUM_POINTS,NUM_CLASSES)
print(model)
```

```
PointNetFull(
  (input_transform_1): InputTransform(
    (t_net): Tnet(
      (fc_1): Conv1d(3, 32, kernel_size=(1,), stride=(1,))
      (bn_1): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_sta
ts=True)
      (fc_2): Conv1d(32, 64, kernel_size=(1,), stride=(1,))
      (bn_2): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_sta
ts=True)
      (fc_3): Conv1d(64, 256, kernel_size=(1,), stride=(1,))
      (bn_3): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
      (mp): MaxPool1d(kernel_size=256, stride=256, padding=0, dilation=1, ceil_mode=F
alse)
      (fc_4): Linear(in_features=1024, out_features=128, bias=True)
      (bn_4): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
      (fc_5): Linear(in_features=128, out_features=64, bias=True)
      (bn_5): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_sta
ts=True)
      (fc_6): Linear(in_features=64, out_features=9, bias=True)
    )
  )
  (fc_1): Conv1d(3, 32, kernel_size=(1,), stride=(1,))
  (bn_1): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
  (fc_2): Conv1d(32, 32, kernel_size=(1,), stride=(1,))
  (bn_2): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
  (fc_3): Conv1d(32, 32, kernel_size=(1,), stride=(1,))
  (bn_3): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
  (fc_4): Conv1d(32, 128, kernel_size=(1,), stride=(1,))
  (bn_4): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
  (fc_5): Conv1d(128, 512, kernel_size=(1,), stride=(1,))
  (bn_5): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
  (mp): MaxPool1d(kernel_size=512, stride=512, padding=0, dilation=1, ceil_mode=Fals
e)
  (fc_6): Linear(in_features=1024, out_features=256, bias=True)
  (bn_6): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
  (fc_7): Linear(in_features=256, out_features=128, bias=True)
  (dropout_1): Dropout(p=0.3, inplace=False)
  (bn_7): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
  (fc_8): Linear(in_features=128, out_features=10, bias=True)
)
```