CS323 Project 5: 3D Deep Learning

Authors: Guocheng Qian and Modar Alfadly and Hasan Hammoud

Due Date: 09 May 2023 @ 11:59 PM

Student Name: David Felipe Alvear Goyes

KAUST ID: 187594

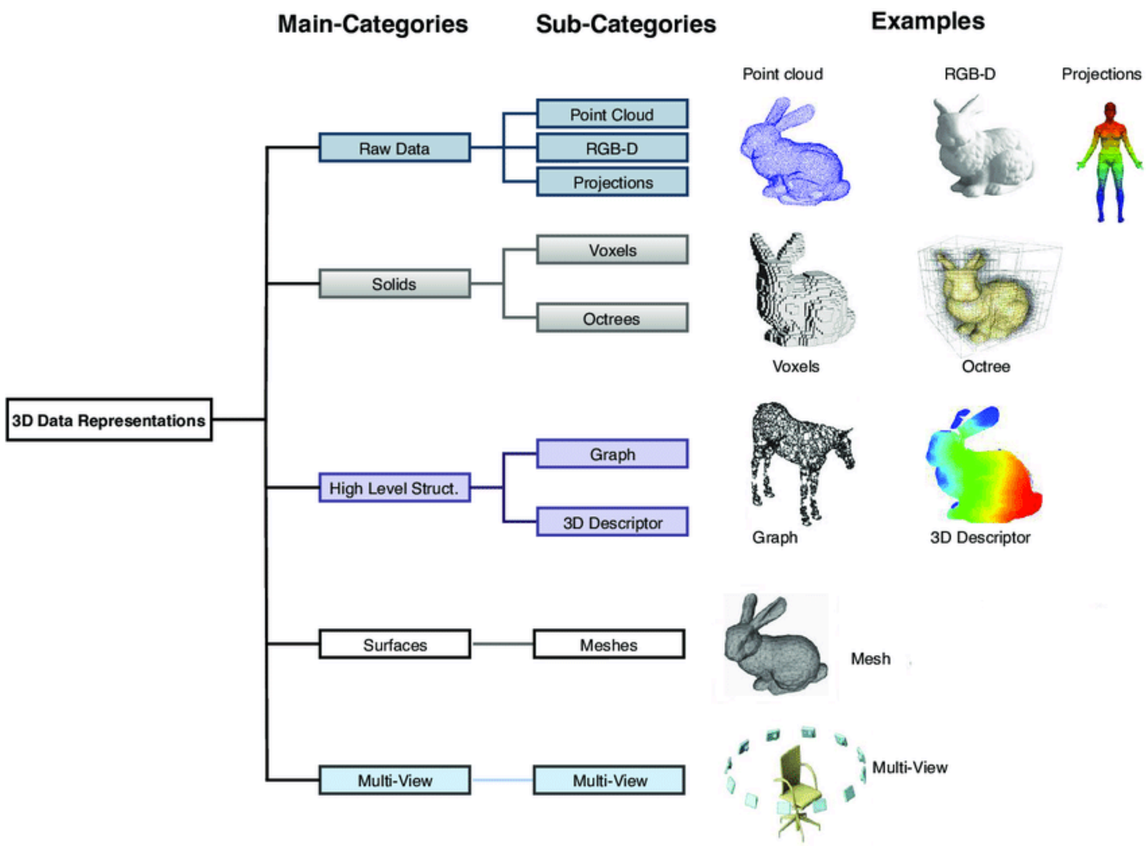
Degree: Electrical and Computer Engineering

Major: Robotics and autonomous systems

To setup a conda environment for this project, just run the following commands:

```
source $(conda info --base)/etc/profile.d/conda.sh
conda create -n cs3232 python=3.9.2 -y
conda activate cs3232

conda install pytorch=1.8.1 torchvision=0.9.1 torchaudio=0.8.1
cudatoolkit=11.1 -c pytorch -c conda-forge -y
conda install jupyter=1.0.0 -y  # to edit this file
conda install matplotlib=3.3.4 -y  # for plotting
conda install tqdm=4.59.0 -y  # for a nice progress bar
conda install h5py=2.10.0 -y  # H5Py for processing HDF5 files
conda install tensorboard=2.4.1 -c conda-forge -y  # to use
tensorboard
pip install pyvista==0.33.2 # for visulizing point cloud
pip install gdown

pip install ftfy regex # for clip
pip install git+https://github.com/openai/CLIP.git # for clip

pip install jupyter_http_over_ws  # for Google Colab
jupyter serverextension enable --py jupyter_http_over_ws  # Google
Colab
```

In the previous projects, you learned about discriminative and generative models and used different neural network architectures; MLPs (for generic features), CNNs (for images), RNNs (for feature sequences), and Transformers (for features sets). In this project, you will learn about Graph Convolutional Networks (GCNs) for tackling graphs (a set of vertices and edges). However, we will use GCNs to do deep learning on point clouds (unstructured non-euclidean data) which is a common representation for geometric data (a set of vertices in 3D).

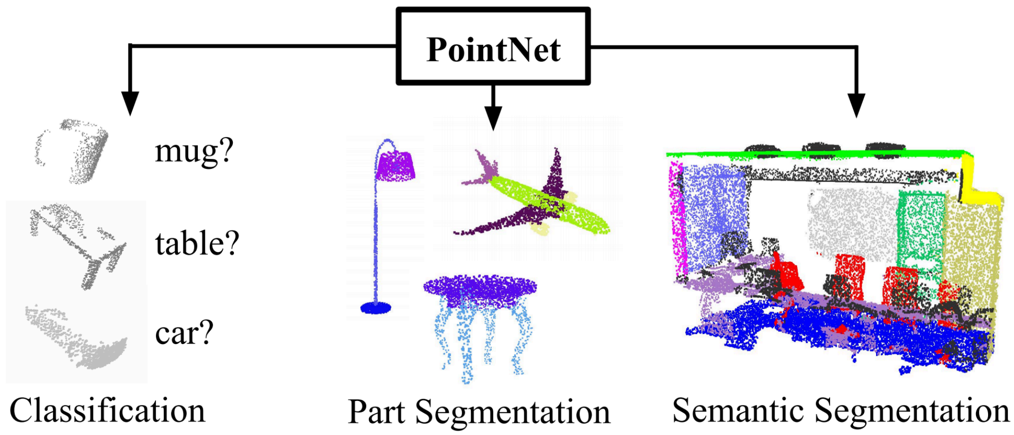


In [26]:
```
import os
import ssl
import math
import enum
import urllib
from pathlib import Path
import glob

from tqdm.notebook import tqdm

import h5py
import numpy as np
import pandas as pd
import sklearn.metrics as metrics

import tensorboard

import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader
from torch.utils.tensorboard import SummaryWriter
# from tensorboardX import SummaryWriter
from torch.utils.collect_env import get_pretty_env_info

from torchvision.datasets.utils import extract_archive, check_integrity
```

Point clouds are a set of unordered points in 3D space. A network that processes N 3D points needs to be invariant to all possible $N!$ permutations of the points. Therefore, using the plain CNN architecture (like ResNet) is not feasible for this task. In this project, we will introduce two permutation-invariant methods to process point clouds:

- PointNet uses shared MLPs to extract per-point features followed by a global pooling operation then a classifier
- DGCNN (a.k.a, EdgeConv, a graph-based method) extracts point neighborhoods and apply convolution-like operatoins on them

You will be asked to implement a simplified version of PointNet and the original DGCNN for point cloud classification on ModelNet40 dataset.



We will do everything with pure PyTorch but using libraries like PyTorch Geometric and Kaolin can make our lives much easier.

Part 1: Setup (2 points)

Task 1: Data Processing (2 points)

ModelNet40 contains $12,311$ meshed CAD models from 40 classes. We work with sampled point clouds from the surfaces of the meshes. Given a point cloud $R^{N\times C}$ (N is the number of points and $C=3$ is (x,y,z)), the goal is to predict which category this point cloud belongs to. From this point forward, we will assume $N=1024$ to be fixed. In addition, a batch of B point clouds will have the shape $[B,C,N]$.

In [27]:
```
def download_and_extract_archive(url, path, md5=None):
    # Works even if the SSL certificate is expired for the link
    path = Path(path)
    path.mkdir(parents=True, exist_ok=True)
    extract_path = path
    file_path = path / Path(url).name
    if not file_path.exists() or not check_integrity(file_path, md5):
        print(f'{file_path} not found or corrupted')
        print(f'downloading from {url}')
        context = ssl.SSLContext()
```

```python
        with urllib.request.urlopen(url, context=context) as response:
            with tqdm(total=response.length) as pbar:
                with open(file_path, 'wb') as file:
                    chunk_size = 1024
                    chunks = iter(lambda: response.read(chunk_size), '')
                    for chunk in chunks:
                        if not chunk:
                            break
                        pbar.update(chunk_size)
                        file.write(chunk)
        extract_archive(str(file_path), str(extract_path))
    return extract_path

def load_data(data_dir, partition, url=None):
    download_and_extract_archive(url, data_dir)
    all_data = []
    all_label = []
    for h5_name in glob.glob(os.path.join(data_dir, 'modelnet40_ply_hdf5_204
        with h5py.File(h5_name, 'r') as f:
            data = f['data'][:].astype('float32')
            label = f['label'][:].astype('int64')
        all_data.append(data)
        all_label.append(label)
    all_data = np.concatenate(all_data, axis=0)
    all_label = np.concatenate(all_label, axis=0).squeeze(-1)
    return all_data, all_label


def vis_points(points, colors=None, labels=None, color_map='Paired'):
    """Visualize a point cloud
    Note about direction in the visualization:  x: horizontal right (red arr
    Args:
        points ([np.array]): [N, 3] numpy array
        colors ([type], optional): [description]. Defaults to None.
    """
    import pyvista as pv
    import numpy as np
    from pyvista import themes
    my_theme = themes.DefaultTheme()
    my_theme.color = 'black'
    my_theme.lighting = True
    my_theme.show_edges = True
    my_theme.edge_color = 'white'
    my_theme.background = 'white'
    pv.set_plot_theme(my_theme)

    assert len(points.shape) < 4, "accept a point cloud with shape [N, 3] as
    if len(points.shape) == 3:
        points = points[0]
        print("only showing the first batch")

    if not isinstance(points, np.ndarray):
        points = points.cpu().numpy()
    if colors is not None and not isinstance(colors, np.ndarray):
        colors = colors.cpu().numpy()
        if len(colors.shape) == 3:
```

```python
            colors = colors[0]

    if colors is None and labels is not None:
        from matplotlib import cm
        if not isinstance(labels, np.ndarray):
            labels = labels.cpu().numpy()

        color_maps = cm.get_cmap(color_map, labels.max() + 1)
        colors = color_maps(labels)

    pointcloud = pv.PolyData(points)
    if colors is not None:
        pointcloud['point_color'] = colors  # point_color, not color.
    pointcloud.plot(rgb=True)


class ModelNet40(Dataset):
    """ModelNet40 dataset"""
    dir_name = 'modelnet40_ply_hdf5_2048'
    md5 = 'c9ab8e6dfb16f67afdab25e155c79e59'
    url = f'https://shapenet.cs.stanford.edu/media/{dir_name}.zip'
    classes = ['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']
    def __init__(self, data_dir='./data/', split='train', transform=None, nu
        data_dir = os.path.join(os.getcwd(), data_dir) if data_dir.startswit
        self.partition = 'train' if split.lower() == 'train' else 'test'  #
        self.data, self.label = load_data(data_dir, self.partition, self.url
        self.num_points = num_points
        print(f'==> sucessfully loaded {self.partition} data')
        self.transform = transform

    def __getitem__(self, index):
        pointcloud = torch.from_numpy(self.data[index][:self.num_points].ast
        label = self.label[index]

        if self.transform is not None:
            pointcloud = self.transform(pointcloud)

        return pointcloud.transpose(1,0).contiguous(), label

    def __len__(self):
        return self.data.shape[0]

    def show(self, item=None):
        pointcloud = self.data[item][:self.num_points]
        vis_points(pointcloud)
```

In [19]:
```
ModelNet40(split='test')#.show(1)
```

```
==> sucessfully loaded test data
```

Out[19]:
```
<__main__.ModelNet40 at 0x7ff304d3f9d0>
```

In [28]:
```
def point_cloud_transform(point_cloud):
    """Transformation function for point clouds

    Args:
        point_cloud: tensor of shape [3, N]
        training: whether in training or testing mode

    Returns:
        point cloud of shape [3, N] (scaled and shifted if training)
    """
    # TODO: vvvvvvvvvvv (1 points)
    # scale then shift the entire point cloud along (x, y, z)
    # both scale and shift are single vectors of size [3]
    # scale is sampled uniformly between [2/3, 3/2]
    # shift is sampled unifromly between [-0.2, 0.2]

    # hint:
    # scale: re-scale a given point cloud
    # shift: translate a given point cloud along x, y, z

    scale = torch.rand(3) * (3/2 - 2/3) + 2/3
```

```python
    shift = torch.rand(3) * (0.4) - 0.2
#     point_cloud = point_cloud * scale.view(3, 1) + shift.view(3, 1)
    point_cloud = point_cloud * scale + shift
    # ^^^^^^^^^^^^^^^^^

    # TODO: vvvvvvvvvvv (0.5 points)
    # shuffle the points just in case the model depends on the order
    rand_idx = torch.randperm(point_cloud.shape[1])
    point_cloud = point_cloud[:, rand_idx]
    # ^^^^^^^^^^^^^^^^^
    return point_cloud
```

In [29]:

```python
# load the train and test
train_set = ModelNet40(split='train',transform = point_cloud_transform )
test_set =  ModelNet40(split='test',transform = None)
```

```
==> sucessfully loaded train data
==> sucessfully loaded test data
```

In [30]:

```python
# TODO: vvvvvvvvvvv (0.5 points)
# create train_loader and test_loader for ModelNet40


device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
print(f'using {device}...')


# set batch size to 32
batch_size = 32
train_loader = DataLoader(train_set,
                         batch_size=batch_size,
                         shuffle=True,
                         num_workers=2)

test_loader = DataLoader(test_set,
                         batch_size=batch_size,
                         shuffle=True,
                         num_workers=2)
# ^^^^^^^^^^^^^^^^^
```

```
using cuda:0...
```

In [23]:

```python
class Phase(enum.Enum):
    TRAINING = TRAIN = enum.auto()
    VALIDATION = VALID = VAL = enum.auto()
    TESTING = TEST = enum.auto()


loaders = {
    Phase.TRAINING: train_loader,
    Phase.VALIDATION: test_loader,
    Phase.TESTING: test_loader,
}
```

Task 2: Training and Evaluation (0 point)

Here is a copy of `ClassificationMetrics` class that we used in previous projects with a single twist; it computes the cross entropy loss.

In [24]:
```
class ClassificationMetrics(nn.Module):
    """Accumulate per-category classification metrics"""
    metrics = ('recall', 'precision', 'f1_score', 'iou')

    def __init__(self, num_classes):
        super().__init__()
        self.criterion = nn.CrossEntropyLoss()
        classes = torch.arange(num_classes)
        zeros = torch.zeros(num_classes, dtype=torch.long)
        self.register_buffer('true_positive', zeros)
        self.register_buffer('false_negative', zeros.clone())
        self.register_buffer('false_positive', zeros.clone())
        self.register_buffer('true_negative', zeros.clone())
        self.register_buffer('classes', classes.unsqueeze(1))
        self.register_buffer('_loss', torch.zeros(()))

    @property
    def loss(self):
        """Average loss"""
        return self._loss / self.total

    def forward(self, logits, targets):
        """Perform the forward pass"""
        logits, targets = logits.flatten(1), targets.flatten()
        loss = self.criterion(logits, targets)
        if self.training:
            self._loss += loss.item() * len(targets)
            self.update(logits.data.argmax(dim=1), targets)
        return loss

    @property
    def count(self):
        """Get the number of samples per-class"""
        return self.true_positive + self.false_negative

    @property
    def frequency(self):
        """Get the per-class frequency"""
        count = self.true_positive + self.false_negative
        return count / count.sum().clamp_min_(1)

    @property
    def total(self):
        """Get the total number of samples"""
        return self.true_positive.sum() + self.false_negative.sum()

    def update(self, pred, true):
        """Update the confusion matrix with the given predictions"""
```

```
        pred, true = pred.data.flatten(), true.data.flatten()
        valid = (true >= 0) & (true < len(self.classes))
        pred_pos = self.classes == pred[valid].unsqueeze(0)
        positive = self.classes == true[valid].unsqueeze(0)
        pred_neg, negative = ~pred_pos, ~positive
        self.true_positive += (pred_pos & positive).sum(dim=1)
        self.false_positive += (pred_pos & negative).sum(dim=1)
        self.false_negative += (pred_neg & positive).sum(dim=1)
        self.true_negative += (pred_neg & negative).sum(dim=1)
        return self

    def reset(self):
        """Reset all running meters"""
        self._loss.zero_()
        self.true_positive.zero_()
        self.false_negative.zero_()
        self.false_positive.zero_()
        self.true_negative.zero_()

    @property
    def accuracy(self):
        """Get the per-class accuracy"""
        den = self.total.clamp_min_(1)
        return (self.true_positive + self.true_negative) / den

    @property
    def recall(self):
        """Get the per-class recall"""
        den = (self.true_positive + self.false_negative).clamp_min_(1)
        return self.true_positive / den

    @property
    def precision(self):
        """Get the per-class precision"""
        den = (self.true_positive + self.false_positive).clamp_min_(1)
        return self.true_positive / den

    @property
    def f1_score(self):
        """Get the per-class F1 score"""
        num = 2 * self.true_positive
        den = (num + self.false_positive + self.false_negative).clamp_min_(1
        return num / den

    @property
    def iou(self):
        """Get the per-class intersection over union"""
        den = self.true_positive + self.false_positive + self.false_negative
        return self.true_positive / den.clamp_min_(1)

    def weighted(self, scores):
        """Compute the weighted sum of per-class metrics"""
        return (self.frequency * scores).sum()

    def __getattr__(self, name):
        """Quick hack to add mean and weighted properties"""
```

```
        if name.startswith('mean_') or name.startswith('weighted_'):
            metric = getattr(self, '_'.join(name.split('_')[1:]))
            if name.startswith('mean_'):
                return metric.mean()
            return self.weighted(metric)
        return super().__getattr__(name)

    def __repr__(self):
        percent = lambda values: (f'{x:.2f}%' for x in values)
        row = lambda *values: ''.join(x.rjust(10) for x in values)
        metrics = torch.stack([getattr(self, m) for m in self.metrics]) * 10
        return '\n'.join([
            f'loss = {self.loss:.5f} (averaged over {self.total} samples)',
            row('', *self.metrics),
            row('mean', *percent(metrics.mean(dim=1))),
            row('weighted', *percent((self.frequency * metrics).sum(dim=1)))
        ])

    def log(self, writer, epoch, prefix=None):
        """Log the results"""
        writer.add_scalar(f'Loss/{prefix}', self.loss, epoch)
        for metric in self.metrics:
            values = getattr(self, metric)
            mean = values.mean()
            weighted = self.weighted(values)
            metric = metric.title()
            writer.add_scalar(f'{metric}/Mean/{prefix}', mean, epoch)
            writer.add_scalar(f'{metric}/Weighted/{prefix}', weighted, epoch
            for i, x in enumerate(values):
                writer.add_scalar(f'Class/{i}/{metric}/{prefix}', x, epoch)
```

Performing one training/evaluation epoch now becomes easy using this class.

In [25]:
```
def one_epoch(phase, model, loader, device, optimizer=None, scheduler=None):
    """Perform one epoch"""
    metrics = None
    training = phase is Phase.TRAINING
    cos_ann_warm = torch.optim.lr_scheduler.CosineAnnealingWarmRestarts
    with torch.set_grad_enabled(training):
        model.train(training)
        for i, (inputs, targets) in enumerate(tqdm(loader)):
            logits = model(inputs.to(device))
            if metrics is None:
                num_classes = logits.shape[1:].numel()
                metrics = ClassificationMetrics(num_classes).to(device)
            loss = metrics(logits.to(device),targets.to(device))
            # ^^^^^^^^^^^^^^^^^
            if training:
                optimizer.zero_grad()
                loss.backward()
                optimizer.step()
                if isinstance(scheduler, cos_ann_warm):
                    epoch = max(scheduler.last_epoch, 0)
                    scheduler.step(epoch + i / len(loader))
    return metrics
```

Here, I implemented for you a generic classifier training function. It supports resuming training from the last best epoch and tensorboard logging. It also supports all learning rate schedulers. This might get overwhelming at some point and as a researcher, you might not have the time to write all of this boilerplate code. This is why, high-level APIs are invented for deep learning frameworks like PyTorch-Lightning.

In [26]:

```
# PyTorch, handling training, validation, early stopping, and checkpoint sav
def train(model, loaders, optimizer, scheduler, device, epochs, log_dir=None
    """Train a classifier model"""
    if log_dir is None:
        epoch = best_value = 0
    else:
        # can we resume from a previouse checkpoint?
        checkpoint_path = Path(log_dir) / 'checkpoint.pt'
        checkpoint_path.parent.mkdir(exist_ok=True, parents=True)
        if checkpoint_path.exists():
            state = torch.load(checkpoint_path, map_location=device)
            epoch = state['epoch']
            best_value = state['best_value']
            model.load_state_dict(state['model'])
            try:
                optimizer.load_state_dict(state['optimizer'])
            except:
                print('could not load the optimizer')
            try:
                scheduler.load_state_dict(state['scheduler'])
            except:
                print('could not load the scheduler')
            if epoch >= epochs:
                tested = Phase.TESTING.name.title() in state['metrics']
                if tested or Phase.TESTING not in loaders:
                    print(f'Already trained for {epoch} epochs!')
                    return
            del state
        else:
            epoch = best_value = 0
        writer = SummaryWriter(log_dir)

    # a function to perform one epoch with logging
    def run_epoch(epoch, phase, metrics=None):
        print(f'Epoch: {epoch} ({phase.name})'.ljust(22) + '#' * 33)
        loader = loaders[phase]
        result = one_epoch(phase, model, loader, device, optimizer, schedule
        if log_dir is not None:
            result.log(writer, epoch, phase.name.title())
        if metrics is not None:
            metrics[phase.name.title()] = result.state_dict()
        print(result)
        return result.loss, result.mean_recall  # commonly known as accuracy

    # start training if necessary
    last_best = 0
    print(get_pretty_env_info())
    print(f'Working on {device}')
```

```python
    print(model)
    print(f'Number of available threads: {torch.get_num_threads()}')
    if epoch < epochs:
        print(f'Training from epoch {epoch + 1} to {epochs}')
    for epoch in range(epoch + 1, epochs + 1):
        metrics = {}
        should_stop = False
        for phase in [Phase.TRAINING, Phase.VALIDATION]:
            loss, current_value = run_epoch(epoch, phase, metrics)
            if phase is Phase.TRAINING:
                if math.isnan(loss) or math.isinf(loss):
                    print(f'Reached invalid loss! {loss}')
                    should_stop = True
                    break
        if should_stop:
            break
        if isinstance(scheduler, torch.optim.lr_scheduler.ReduceLROnPlateau)
            scheduler.step(current_value)
        else:
            scheduler.step()
        if current_value >= best_value:
            last_best, best_value = 0, current_value
            if log_dir is not None:
                state = {
                    'epoch': epoch,
                    'best_value': best_value,
                    'model': model.state_dict(),
                    'optimizer': optimizer.state_dict(),
                    'scheduler': scheduler.state_dict(),
                    'metrics': metrics,
                }
                torch.save(state, checkpoint_path)
                del state
        else:
            last_best += 1
        if last_best > getattr(scheduler, 'patience', 10) + 4:
            print(f'Early stopping! (waited {last_best} epochs)')
            break

    # compute testing metrics
    if Phase.TESTING in loaders:
        metrics = None
        phase = Phase.TESTING
        if log_dir is not None:
            best_state = torch.load(checkpoint_path, map_location=device)
            epoch = best_state['epoch']
            metrics = best_state['metrics']
            model.load_state_dict(best_state['model'])
        run_epoch(epoch, phase, metrics)
        if log_dir is not None:
            torch.save(best_state, checkpoint_path)
```

Let's monitor our training in real-time using tensorboard.

In [27]:

```
# TODO: vvvvvvvvvvv (0 points)
# start tensorboard here
# https://www.tensorflow.org/tensorboard/tensorboard_in_notebooks
# you should be able to see tensorboard (No dashboards) here after tunning t

# no need to write any code here, just to make sure the tensorboard works fi
log_dir = Path('./runs')
log_dir.mkdir(exist_ok=True)
tensorboard.notebook.start(f'--logdir={log_dir} --bind_all')

tensorboard.notebook.list()
# ^^^^^^^^^^^^^^^^^
```

```
Reusing TensorBoard on port 6006 (pid 2916190), started 3 days, 4:21:44 ago.
(Use '!kill 2916190' to kill it.)
```

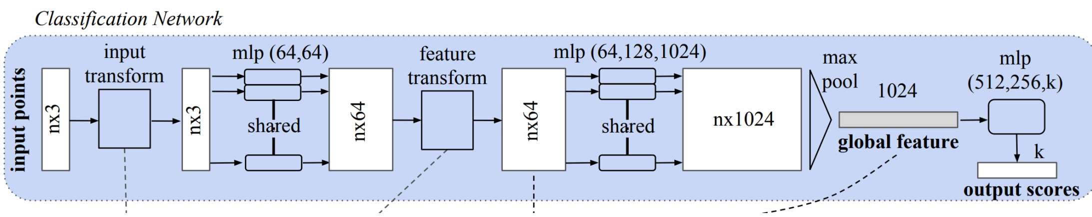
Index of /

Name	Size	Date Modified
.vol/		5/7/24, 10:01:44 AM
Applications/		9/15/24, 9:22:05 AM
bin/		5/7/24, 10:01:44 AM
cores/		3/3/21, 1:24:44 PM
dev/		9/8/24, 9:33:19 AM
etc/		6/26/24, 7:39:58 PM
home/		9/8/24, 9:33:55 AM
Library/		6/26/24, 7:41:24 PM
opt/		5/17/23, 1:51:46 PM
private/		9/8/24, 9:33:35 AM
sbin/		5/7/24, 10:01:44 AM
System/		5/7/24, 10:01:44 AM
tmp/		9/15/24, 10:58:56 AM
Users/		6/26/24, 7:38:33 PM
usr/		5/7/24, 10:01:44 AM
var/		6/26/24, 7:39:48 PM
Volumes/		9/15/24, 10:41:30 AM
.file	0 B	5/7/24, 10:01:44 AM

```
Known TensorBoard instances:
  - port 6006: logdir runs (started 3 days, 4:21:44 ago; pid 2916190)
```

Part 2: PointNet (3 points)

PointNet is the first influential work that process on point cloud direclty without projecting the points into voxels or images. Read PointNet and make sure your understand PointNet. In this part, we will implement the PointNet classifier as in the figure below (Section 4.2 and Appendix C).



Note: we are going to implement a simplified PointNet; without T-Net (input/feature transform) as they are not that critical.

We will build the basic modules first which you **should use them** to build the simplified PointNet architecture.

In [28]:
```
def activation_layer(activation='relu', inplace=True, slope=0.2, prelu=1):
    """Get an activation function layer given the name"""
    activation = activation.lower()
    if activation in ('none', 'identity'):
        layer = nn.Identity()
    elif activation == 'relu':
        layer = nn.ReLU(inplace)
    elif activation == 'leakyrelu':
        layer = nn.LeakyReLU(slope, inplace)
    elif activation == 'prelu':
        layer = nn.PReLU(num_parameters=prelu, init=slope)
    else:
        raise ValueError(f'unknown activation layer [{activation}]')
    return layer

def mlp(*channels, bias=True, shared=False, norm=True, dropout=0, **kwargs):
    """Get a Multi-Layer Perceptron (MLP)

    if shared, it is implemented as Conv1d with kernel_size=1
    the input is a point cloud and the same linear layer is applied
    on each point independently. This can be in theory implemented
    as just Linear layer if the point cloud shape was [B, N, C]
    but we decided to implement it this way as an introduction to Part 2

    Args:
        channels: layer dimensions (e.g. [3, 64, 40] gives 2 layers)
        bias: whether to use bias or not
        shared: whether the input is a point cloud or global feature
        norm: whether to use batch norm or not
        dropout: the drop probability of dropout
        kwargs: options for the activation function

    Returns:
        nn.Sequential of the MLP's blocks
```

```
    """
    modules = []
    for in_channels, out_channels in zip(channels, channels[1:]):
        block = []
        if norm:
            bias = False  # don't use bias if you are using batch norm
        if shared:
            linear = nn.Conv1d(in_channels, out_channels, 1, bias=bias)
        else:
            linear = nn.Linear(in_channels, out_channels, bias=bias)
        block.append(linear)

        # TODO: vvvvvvvvvv (0.5 points)
        # append normalization, activation layer, and dropout layer to the b
        if norm:
            block.append(nn.BatchNorm1d(out_channels))
        # activation layer
        block.append(activation_layer(**kwargs))
        if dropout > 0:
            block.append(nn.Dropout(dropout))
        modules.append(nn.Sequential(*block))
    return nn.Sequential(*modules)
```

In [29]:

```
class SimplePointNet(nn.Module):
    def __init__(self, num_classes=40):
        super().__init__()
        # TODO: vvvvvvvvvvv (1 points)
        # build the PointNet w/o any transformation (T-Net)
        # no input transformation, no feature transformation

        # shared MLPs (per-point features)
        # input: [B, 3, N]
        # layers: 3 -> 64 -> 128 -> 1024
        # output: [B, 1024, N]
        # hint: using the mlp block to build
        self.shared = mlp(3,64,128,1024, shared=True, activation='relu', nor
#         self.shared = mlp([3,64,128,1024], shared=True, activation='relu')

        self.pool = nn.Sequential(
                    nn.MaxPool1d(1024),  # pool over the points
                    nn.Flatten(1),
                    )

        # MLP classifier
        # input: [B, 1024]
        # layers: 1024 -> 512 -> 256 -> num_classes
        # output: [B, num_classes]
        # there is a dropout somewhere here
        # hint: using the mlp block with shared to False.
        # hint: using sequential
        # hint: be careful about dropout, normalziation, and activation laye
#         self.classifier = mlp([1024, 512, 256, num_classes],
#                               dropout=0.5, activation='leakyrelu', norm=Tr
        self.classifier = nn.Sequential(
            mlp(1024, 512, dropout=0, activation='relu', norm=True),
```

```
            mlp(512, 256, dropout=0.7, activation='relu', norm=False),
            nn.Linear(256, num_classes),
        )
        # ^^^^^^^^^^^^^^^^^

    def forward(self, point_cloud):
        # TODO: vvvvvvvvvvv (0.5 points)
        # perform the forward pass
        # pass through the shared mlp
        pc_shared = self.shared(point_cloud)
        # pass through the pool layer
        pc_pool = self.pool(pc_shared)
        # pass to the classifier mlp
        pc_classifier = self.classifier(pc_pool)
        return pc_classifier
        # ^^^^^^^^^^^^^^^^^
```

In [30]:

```
import torch.optim as optim

model = SimplePointNet().to(device)

# TODO: vvvvvvvvvv (1 point)
# use the same optimizer and learning scheduler as suggested in the paper
# run the training for 10 epochs
# goal: best weighted test recall should be over 50%
epochs = 10
log_dir = './runs/PointNet/experiment_1'
optimizer = optim.Adam(model.parameters(), lr=0.001, betas=(0.9, 0.999))
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=10, gamma=0.8)
# ^^^^^^^^^^^^^^^^^

train(model, loaders, optimizer, scheduler, device, epochs, log_dir)
del model, optimizer, scheduler
```

```
PyTorch version: 2.0.0+cu117
Is debug build: False
CUDA used to build PyTorch: 11.7
ROCM used to build PyTorch: N/A

OS: Ubuntu 20.04.6 LTS (x86_64)
GCC version: (Ubuntu 9.4.0-1ubuntu1~20.04.1) 9.4.0
Clang version: Could not collect
CMake version: version 3.26.3
Libc version: glibc-2.31

Python version: 3.8.10 (default, Nov 14 2022, 12:59:47)  [GCC 9.4.0] (64-bit
runtime)
Python platform: Linux-5.15.0-69-generic-x86_64-with-glibc2.29
Is CUDA available: True
CUDA runtime version: 11.1.105
CUDA_MODULE_LOADING set to: LAZY
GPU models and configuration:
GPU 0: NVIDIA GeForce RTX 3080
GPU 1: NVIDIA GeForce RTX 3080

Nvidia driver version: 470.182.03
cuDNN version: Could not collect
HIP runtime version: N/A
MIOpen runtime version: N/A
Is XNNPACK available: True

CPU:
Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Byte Order:                      Little Endian
Address sizes:                   43 bits physical, 48 bits virtual
CPU(s):                          64
On-line CPU(s) list:             0-63
Thread(s) per core:              2
Core(s) per socket:              32
Socket(s):                       1
NUMA node(s):                    1
Vendor ID:                       AuthenticAMD
CPU family:                      23
Model:                           49
Model name:                      AMD Ryzen Threadripper 3970X 32-Core Proces
sor
Stepping:                        0
Frequency boost:                 enabled
CPU MHz:                         2200.000
CPU max MHz:                     3700.0000
CPU min MHz:                     2200.0000
BogoMIPS:                        7400.02
Virtualization:                  AMD-V
L1d cache:                       1 MiB
L1i cache:                       1 MiB
L2 cache:                        16 MiB
L3 cache:                        128 MiB
NUMA node0 CPU(s):               0-63
Vulnerability Itlb multihit:     Not affected
```

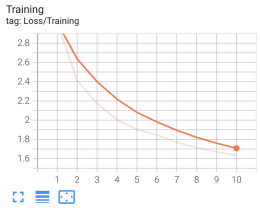
```
Vulnerability L1tf:              Not affected
Vulnerability Mds:               Not affected
Vulnerability Meltdown:          Not affected
Vulnerability Mmio stale data:   Not affected
Vulnerability Retbleed:          Mitigation; untrained return thunk; SMT ena
bled with STIBP protection
Vulnerability Spec store bypass: Mitigation; Speculative Store Bypass disabl
ed via prctl and seccomp
Vulnerability Spectre v1:        Mitigation; usercopy/swapgs barriers and __
user pointer sanitization
Vulnerability Spectre v2:        Mitigation; Retpolines, IBPB conditional, S
TIBP always-on, RSB filling, PBRSB-eIBRS Not affected
Vulnerability Srbds:             Not affected
Vulnerability Tsx async abort:   Not affected
Flags:                           fpu vme de pse tsc msr pae mce cx8 apic sep
mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ht syscall nx mmxext f
xsr_opt pdpe1gb rdtscp lm constant_tsc rep_good nopl nonstop_tsc cpuid extd_
apicid aperfmperf rapl pni pclmulqdq monitor ssse3 fma cx16 sse4_1 sse4_2 mo
vbe popcnt aes xsave avx f16c rdrand lahf_lm cmp_legacy svm extapic cr8_lega
cy abm sse4a misalignsse 3dnowprefetch osvw ibs skinit wdt tce topoext perfc
tr_core perfctr_nb bpext perfctr_llc mwaitx cpb cat_l3 cdp_l3 hw_pstate ssbd
mba ibpb stibp vmmcall fsgsbase bmi1 avx2 smep bmi2 cqm rdt_a rdseed adx sma
p clflushopt clwb sha_ni xsaveopt xsavec xgetbv1 xsaves cqm_llc cqm_occup_ll
c cqm_mbm_total cqm_mbm_local clzero irperf xsaveerptr rdpru wbnoinvd arat n
pt lbrv svm_lock nrip_save tsc_scale vmcb_clean flushbyasid decodeassists pa
usefilter pfthreshold avic v_vmsave_vmload vgif v_spec_ctrl umip rdpid overf
low_recov succor smca sme sev sev_es

Versions of relevant libraries:
[pip3] numpy==1.18.5
[pip3] torch==2.0.0
[pip3] torchsummary==1.5.1
[pip3] torchvision==0.15.1
[pip3] torchviz==0.0.2
[conda] blas                      1.0                         mkl
[conda] mkl                       2021.4.0            h06a4308_640
[conda] mkl-service               2.4.0           py310h7f8727e_0
[conda] mkl_fft                   1.3.1           py310hd6ae3a3_0
[conda] mkl_random                1.2.2           py310h00e6091_0
[conda] numpy                     1.23.5          py310hd5efca6_0
[conda] numpy-base                1.23.5          py310h8e6c178_0
Working on cuda:0
SimplePointNet(
  (shared): Sequential(
    (0): Sequential(
      (0): Conv1d(3, 64, kernel_size=(1,), stride=(1,), bias=False)
      (1): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_runni
ng_stats=True)
      (2): ReLU(inplace=True)
    )
    (1): Sequential(
      (0): Conv1d(64, 128, kernel_size=(1,), stride=(1,), bias=False)
      (1): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_stats=True)
      (2): ReLU(inplace=True)
    )
```

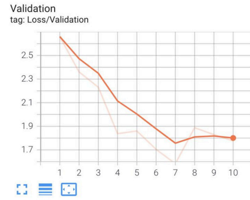
```
    (2): Sequential(
      (0): Conv1d(128, 1024, kernel_size=(1,), stride=(1,), bias=False)
      (1): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (2): ReLU(inplace=True)
    )
  )
  (pool): Sequential(
    (0): MaxPool1d(kernel_size=1024, stride=1024, padding=0, dilation=1, cei
l_mode=False)
    (1): Flatten(start_dim=1, end_dim=-1)
  )
  (classifier): Sequential(
    (0): Sequential(
      (0): Sequential(
        (0): Linear(in_features=1024, out_features=512, bias=False)
        (1): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
        (2): ReLU(inplace=True)
      )
    )
    (1): Sequential(
      (0): Sequential(
        (0): Linear(in_features=512, out_features=256, bias=True)
        (1): ReLU(inplace=True)
        (2): Dropout(p=0.7, inplace=False)
      )
    )
    (2): Linear(in_features=256, out_features=40, bias=True)
  )
)
Number of available threads: 32
Training from epoch 1 to 10
Epoch: 1 (TRAINING)   #################################
  0%|          | 0/308 [00:00<?, ?it/s]
loss = 3.01290 (averaged over 9840 samples)
              recall precision  f1_score       iou
      mean     9.99%    12.27%     8.33%     4.81%
  weighted    21.42%    15.79%    15.70%     9.36%
Epoch: 1 (VALIDATION) #################################
  0%|          | 0/78 [00:00<?, ?it/s]
loss = 2.66002 (averaged over 2468 samples)
              recall precision  f1_score       iou
      mean    16.48%    14.62%    11.79%     8.44%
  weighted    25.24%    22.36%    18.09%    13.08%
Epoch: 2 (TRAINING)   #################################
  0%|          | 0/308 [00:00<?, ?it/s]
loss = 2.40987 (averaged over 9840 samples)
              recall precision  f1_score       iou
      mean    19.31%    23.49%    17.94%    11.21%
  weighted    33.43%    29.17%    29.03%    19.28%
Epoch: 2 (VALIDATION) #################################
  0%|          | 0/78 [00:00<?, ?it/s]
```

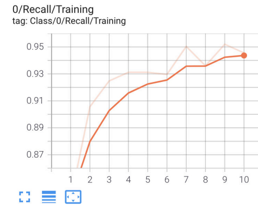
```
loss = 2.36054 (averaged over 2468 samples)
              recall precision  f1_score       iou
      mean    23.12%    28.10%    20.19%    13.68%
  weighted    33.27%    33.94%    27.53%    18.95%
Epoch: 3 (TRAINING)   #################################
  0%|          | 0/308 [00:00<?, ?it/s]
loss = 2.17258 (averaged over 9840 samples)
              recall precision  f1_score       iou
      mean    24.95%    27.08%    23.71%    15.24%
  weighted    39.07%    34.81%    35.40%    24.31%
Epoch: 3 (VALIDATION) #################################
  0%|          | 0/78 [00:00<?, ?it/s]
loss = 2.23098 (averaged over 2468 samples)
              recall precision  f1_score       iou
      mean    29.39%    38.21%    25.50%    17.97%
  weighted    37.84%    39.78%    32.03%    22.86%
Epoch: 4 (TRAINING)   #################################
  0%|          | 0/308 [00:00<?, ?it/s]
loss = 2.00435 (averaged over 9840 samples)
              recall precision  f1_score       iou
      mean    29.43%    31.39%    28.60%    18.82%
  weighted    43.69%    39.61%    40.33%    28.30%
Epoch: 4 (VALIDATION) #################################
  0%|          | 0/78 [00:00<?, ?it/s]
loss = 1.83770 (averaged over 2468 samples)
              recall precision  f1_score       iou
      mean    36.62%    40.01%    33.84%    24.10%
  weighted    45.14%    45.31%    40.90%    29.58%
Epoch: 5 (TRAINING)   #################################
  0%|          | 0/308 [00:00<?, ?it/s]
loss = 1.90152 (averaged over 9840 samples)
              recall precision  f1_score       iou
      mean    31.13%    33.18%    30.40%    20.24%
  weighted    45.92%    41.89%    42.64%    30.33%
Epoch: 5 (VALIDATION) #################################
  0%|          | 0/78 [00:00<?, ?it/s]
loss = 1.86018 (averaged over 2468 samples)
              recall precision  f1_score       iou
      mean    35.25%    38.76%    30.92%    22.74%
  weighted    45.18%    41.76%    37.75%    27.58%
Epoch: 6 (TRAINING)   #################################
  0%|          | 0/308 [00:00<?, ?it/s]
loss = 1.84660 (averaged over 9840 samples)
              recall precision  f1_score       iou
      mean    32.64%    33.77%    31.66%    21.39%
  weighted    47.40%    43.22%    44.24%    31.86%
Epoch: 6 (VALIDATION) #################################
  0%|          | 0/78 [00:00<?, ?it/s]
loss = 1.70477 (averaged over 2468 samples)
              recall precision  f1_score       iou
      mean    35.86%    50.24%    34.19%    24.68%
  weighted    49.59%    58.25%    45.03%    33.61%
Epoch: 7 (TRAINING)   #################################
  0%|          | 0/308 [00:00<?, ?it/s]
```

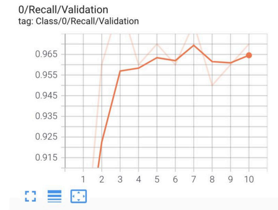
```
loss = 1.76811 (averaged over 9840 samples)
              recall precision  f1_score       iou
      mean    34.29%    35.50%    33.37%    22.75%
  weighted    49.05%    45.14%    46.02%    33.55%
Epoch: 7 (VALIDATION) #################################
  0%|          | 0/78 [00:00<?, ?it/s]
loss = 1.58292 (averaged over 2468 samples)
              recall precision  f1_score       iou
      mean    44.96%    48.25%    41.82%    31.17%
  weighted    53.57%    52.85%    48.55%    37.30%
Epoch: 8 (TRAINING)   #################################
  0%|          | 0/308 [00:00<?, ?it/s]
loss = 1.71441 (averaged over 9840 samples)
              recall precision  f1_score       iou
      mean    35.58%    36.39%    34.71%    23.89%
  weighted    50.65%    46.42%    47.54%    34.89%
Epoch: 8 (VALIDATION) #################################
  0%|          | 0/78 [00:00<?, ?it/s]
loss = 1.88795 (averaged over 2468 samples)
              recall precision  f1_score       iou
      mean    38.46%    44.25%    35.89%    26.50%
  weighted    44.29%    51.63%    42.72%    32.22%
Epoch: 9 (TRAINING)   #################################
  0%|          | 0/308 [00:00<?, ?it/s]
loss = 1.67045 (averaged over 9840 samples)
              recall precision  f1_score       iou
      mean    36.63%    37.83%    35.91%    24.82%
  weighted    51.46%    47.96%    48.73%    36.04%
Epoch: 9 (VALIDATION) #################################
  0%|          | 0/78 [00:00<?, ?it/s]
loss = 1.82907 (averaged over 2468 samples)
              recall precision  f1_score       iou
      mean    39.02%    48.10%    37.85%    28.40%
  weighted    47.16%    51.89%    43.31%    33.19%
Epoch: 10 (TRAINING)  #################################
  0%|          | 0/308 [00:00<?, ?it/s]
loss = 1.63147 (averaged over 9840 samples)
              recall precision  f1_score       iou
      mean    37.80%    39.54%    37.21%    25.87%
  weighted    52.58%    49.14%    49.79%    37.00%
Epoch: 10 (VALIDATION)#################################
  0%|          | 0/78 [00:00<?, ?it/s]
loss = 1.77609 (averaged over 2468 samples)
              recall precision  f1_score       iou
      mean    39.45%    48.10%    37.66%    27.48%
  weighted    47.49%    54.84%    43.92%    32.47%
Epoch: 7 (TESTING)    #################################
  0%|          | 0/78 [00:00<?, ?it/s]
loss = 1.58292 (averaged over 2468 samples)
              recall precision  f1_score       iou
      mean    44.96%    48.25%    41.82%    31.17%
  weighted    53.57%    52.85%    48.55%    37.30%
```

TODO (1.0 points): Attach a screenshot of the Loss and Recall tabs from TensorBoard in a text cell below:









Part 3: Graph Convolutional Networks (3.5 points)

The problem with PointNet is that no point can know about any other point until they reach the global pooling bottleneck. The shared MLPs prevent any comunication of information between the points. A better approach can be formulated in a message passing framework. In particular, we start thinking about our point cloud as a directed graph, where every point is a vertex (node) in the graph with edges connecting to the *neighbors* of the point. The way we define this neighborhood relationship is up to us but usually we just use the k-nearest neighbors (kNNs). Once we have represented the point cloud as a graph, we can just aggregate the features of the neighbors per-point. With this, we have the most basic building block of graph convolutional networks (GCNs):

$$\mathbf{x}_i' = \gamma_{\mathbf{\Theta}} \left(\mathbf{x}_i, \square_{j \in \mathcal{N}(i)} \phi_{\Theta} \left(\mathbf{x}_i, \mathbf{x}_j, \mathbf{e}_{ji} \right) \right)$$

- where $\square$ denotes a differentiable, permutation invariant aggregation function (e.g., sum, mean or max)
- $\gamma_{\mathbf{\Theta}}$ and ϕ_{Θ} denote permutation invariant differentiable functions such as MLPs and 1×1 convolutions
- $\mathbf{x}_i, \mathbf{x}_j, \mathbf{e}_{ji}$ is the center node, the neighbor node, and the edge from node j to node i
- $\mathcal{N}(i)$ is the neighborhood of node i
- $\mathbf{x}_i'$ is the output of node i

Task 1: Edge Convolution (1.5 points)

Deep Graph Convolutional Neural Networks (DGCNN) have defined an example of such operation and called it EdgeConv:

$$\mathbf{x}_i' = \max_{j \in \mathcal{N}(i)} h_{\mathbf{\Theta}} \left(\mathrm{concat} \left[\mathbf{x}_i, \mathbf{x}_j - \mathbf{x}_i \right] \right)$$

If we want to implement this, we can immediately see that the input is a point cloud $\mathbf{x}_i$ of shape `[B, C, N]` with the K neighbors per point given by their indices $\mathcal{N}(i)$ of shape `[B, N, K]` and the output is the updated point cloud $\mathbf{x}_i'$ of shape `[B, C', N]`. The only tricky part here is the edges $\mathbf{x}_j - \mathbf{x}_i$ which have the shape `[B, C, N, K]`. After the concatenation with $\mathbf{x}_i$, they become `[B, C + C, N, K]`. Then, $h_{\mathbf{\Theta}}$ will act on the channels producing `[B, C', N, K]`. Finally, the $\max$ is done over the neighbors which will give our desired output `[B, C', N]`.

Note: in a similar spirit to what we did in PointNet, we can leverage Conv2d with 1×1 kernel size here.

In [32]:
```
def cnn(*channels, bias=True, norm=True, dropout=0, **kwargs):
    """Get a 2D Convolutional Neural Network (CNN)

    Args:
        channels: layer dimensions (e.g. [3, 64, 40] gives 2 layers)
        bias: whether to use bias or not
        norm: whether to use batch norm or not
        dropout: the drop probability of dropout
        kwargs: options for the activation function

    Returns:
        nn.Sequential of the CNN's blocks
    """
    # TODO: vvvvvvvvvvv (0.5 points)
    # build this in exactly similar manner to what we did with mlp
    # the only difference here is that we use Conv2d and BatchNorm2d
    # and it is always shared (we don't have nonshared mode)

    modules = []
    for in_channels, out_channels in zip(channels, channels[1:]):
        block = []
        if norm:
            bias = False  # don't use bias if you are using batch norm
        linear = nn.Conv2d(in_channels, out_channels, 1, bias=bias)
        block.append(linear)

        # TODO: vvvvvvvvvv (0.5 points)
        # append normalization, activation layer, and dropout layer to the b
        if norm:
            block.append(nn.BatchNorm2d(out_channels))
        # activation layer
        block.append(activation_layer(**kwargs))
        if dropout > 0:
            block.append(nn.Dropout(dropout))
        modules.append(nn.Sequential(*block))
    return nn.Sequential(*modules)


# test
cnn(4 * 2, 12)
# ^^^^^^^^^^^^^^^^^
```

Out[32]:
```
Sequential(
  (0): Sequential(
    (0): Conv2d(8, 12, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (1): BatchNorm2d(12, eps=1e-05, momentum=0.1, affine=True, track_runnin
g_stats=True)
    (2): ReLU(inplace=True)
  )
)
```

In [35]:
```python
def gather_features(features, indices, sparse_grad=False):
    """Gather the features specified by indices

    Args:
        features: tensor of shape [B, C, N]
        indices: long tensor of shape [B, N, K]
        sparse_grad: whether to use a sparse tensor for the gradient

    Returns:
        gathered_features [B, C, N, K]
    """
    # unsqueeze the tensors preparing for broadcasting
    features, indices = features.unsqueeze(-1), indices.unsqueeze(-3)
    # Broadcast the tensors to have the same shape
    features, indices = torch.broadcast_tensors(features, indices)
    # return and gather the features to have tensor [B, C, N, K] K neighbors
    return features.gather(dim=-2, index=indices, sparse_grad=sparse_grad)
```

In [38]:
```python
class EdgeConv(nn.Module):
    """Static Edge Convolutional Layer"""
    def __init__(self, in_channels, out_channels, pool='max', **kwargs):
        super().__init__()
        pool = pool.lower()
        self.shared = cnn(in_channels * 2, out_channels, **kwargs)
        if pool == 'max':
            self.pool = lambda x: torch.max(x, dim=-1, keepdim=False)[0]
        elif pool in ['mean', 'avg']:
            self.pool = lambda x: torch.mean(x, dim=-1, keepdim=False)
        elif pool == 'sum':
            self.pool = lambda x: torch.sum(x, dim=-1, keepdim=False)
        else:
            raise NotImplementedError(f'reduction {self.reduction} not imple

    def forward(self, point_cloud, edge_index):
        """Perform the forward pass

        Args:
            point_cloud: tensor of shape [B, C, N]
            edge_index: tensor of shape [B, N, K]

        Returns:
            output point cloud of shape [B, C', N]
        """
        # TODO: vvvvvvvvvvv (1 points)
        # Prepare x_i and x_j to do the forward pass correctly
```

```
        x_j = gather_features(point_cloud,edge_index) # [B,C,N,K]
        # expand point_cloud tensor to have the dimensions of x_j
        x_i = point_cloud.unsqueeze(-1).expand(-1, -1, -1, x_j.size(-1))
        # concatenate the tensors x_i, x_j along the channel dimension
        x_cat = torch.cat((x_i, x_j), dim=1) # [B,C+C,N,K]
        # pass the concatenated vector through the shared conv layer
        x_shared = self.shared(x_cat) # [B,C',N,K]
        # pass the tensor through the pooling operation to have the dimensio
        return self.pool(x_shared) # [B,C',N]


# TEST your EdgeConv here
point_cloud = torch.randn(7, 3, 1024)  # [B, C, N]
edge_index = torch.randint(1024, (7, 1024, 20))  # [B, N, K]
print(EdgeConv(3, 64)(point_cloud, edge_index).shape)  # [B, C', N]
del point_cloud, edge_index
```

```
torch.Size([7, 64, 1024])
```

Task 2: Dynamic Edge Convolution (0.5 points)

The dynamic edge convolutional layer is simply a static EdgeConv with kNN as the neighborhood function. Dynamic Edge Conv queires neighbors before passing the point cloud into a static EdgeConv.

In [41]:
```
def get_neighbors(num_neighbors, features, neighbors=None, p_norm=2,
                  farthest=False, ordered=False):
    """Get the distances and indices to a fixed number of neighbors

    https://gist.github.com/ModarTensai/60fe0d0e3536adc28778448419908f47

    Args:
        num_neighbors: number of neighbors to consider
        features: query points which we need their neighbors [B, C, N]
        neighbors: set of support points (`features` if None) [B, C, M]
        p_norm: distances are computed based on L_p norm
        farthest: whether to get the farthest or the nearest neighbors
        ordered: distance sorted (descending if `farthest`)

    Returns:
        (distances, indices) both of shape [B, N, `num_neighbors`]
    """
    features = features.movedim(-1, -2)
    if neighbors is None:
        neighbors = features
    else:
        neighbors = neighbors.movedim(-1, -2)
    # Compute the pairwise distances between features and neighbors using th
    pairs = torch.cdist(features, neighbors, p_norm)
    # Get the top-k nearest/farthest neighbors' distances and indices
    return pairs.topk(num_neighbors, dim=-1, largest=farthest, sorted=ordere

# TODO: test knn
pointcloud = torch.randn(7, 3, 1024)
```

```
support = torch.randn(7, 3, 2048)
get_neighbors(20, pointcloud, support).indices.shape
```

Out[41]: torch.Size([7, 1024, 20])

In [59]:
```
class DynEdgeConv(EdgeConv):
    """Dynamic Edge Convolutional Layer"""
    def __init__(self, in_channels, out_channels, num_neighbors=20,
                 pool='max', **kwargs):
        self.num_neighbors = num_neighbors
        super().__init__(in_channels, out_channels, pool=pool, **kwargs)

    def forward(self, point_cloud):
        # TODO: vvvvvvvvvvv (0.5 points)
        # perform the forward pass of DynEdgeConv
        knn_out = get_neighbors(self.num_neighbors, point_cloud)
#         print(f'Knn output: {knn_out.indices.shape}') # [7, 1024, 20]
        return super().forward(point_cloud, knn_out.indices) # inherit forwa
        # ^^^^^^^^^^^^^^^^^^


DynEdgeConv(3, 64)(torch.randn(7, 3, 1024)).shape
```

Out[59]: torch.Size([7, 64, 1024])

Task 3: Simple DGCNN (1.5 points)

Instead of reimplementing DGCNN, lets simply replace the shared MLP layers in the backbone of PointNet with DynEdgeConv.

In [60]:
```
class SimpleDGCNN(SimplePointNet):
    def __init__(self, num_classes=40):
        super().__init__(num_classes)
        # TODO: vvvvvvvvvvv (1 points)
        # replace shared MLPs with DynEdgeConv
        # input: [B, 3, N]
        # layers: 3 -> 64 -> 128 -> 1024
        # output: [B, 1024, N]
        self.shared = nn.Sequential(
            DynEdgeConv(3,64),
            DynEdgeConv(64,128),
            DynEdgeConv(128,1024)
        )
        # ^^^^^^^^^^^^^^^^^
        # do not write anything else. SimpleDGCNN inherite most of the parts

out = SimpleDGCNN()(torch.randn(7, 3, 1024))
print(out.shape)
```

torch.Size([7, 40])

In [61]:
```
# set device to cuda
```

```
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
print(f'using {device}...')


# create train_loader and test_loader for ModelNet40

# just copy from Part 1.
# you might need a smaller batch size like B=8
batch_size = 8
train_loader = DataLoader(train_set,
                          batch_size=batch_size,
                          shuffle=True,
                          num_workers=2)

test_loader = DataLoader(test_set,
                         batch_size=batch_size,
                         shuffle=True,
                         num_workers=2)

class Phase(enum.Enum):
    TRAINING = TRAIN = enum.auto()
    VALIDATION = VALID = VAL = enum.auto()
    TESTING = TEST = enum.auto()


loaders = {
    Phase.TRAINING: train_loader,
    Phase.VALIDATION: test_loader,
    Phase.TESTING: test_loader,
}
# ^^^^^^^^^^^^^^^^^
```

```
using cuda:0...
```

In [62]:

```
# decrease the batch size if you encounter out of memory issue
model = SimpleDGCNN().to(device)

# TODO: vvvvvvvvvvv (0.5 points)
# use the same optimizer and learning scheduler as we did in PointNet
# run the training for at least 10 epochs
# soft goal: best weighted test recall should be over 50%
epochs = 10
log_dir = './runs/DGCNN/experiment_1'
optimizer = optim.Adam(model.parameters(), lr=0.001, betas=(0.9, 0.999))
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=20, gamma=0.75)
# ^^^^^^^^^^^^^^^^^

train(model, loaders, optimizer, scheduler, device, epochs, log_dir)
del model, optimizer, scheduler
```

```
PyTorch version: 2.0.0+cu117
Is debug build: False
CUDA used to build PyTorch: 11.7
ROCM used to build PyTorch: N/A

OS: Ubuntu 20.04.6 LTS (x86_64)
GCC version: (Ubuntu 9.4.0-1ubuntu1~20.04.1) 9.4.0
Clang version: Could not collect
CMake version: version 3.26.3
Libc version: glibc-2.31

Python version: 3.8.10 (default, Nov 14 2022, 12:59:47)  [GCC 9.4.0] (64-bit
runtime)
Python platform: Linux-5.15.0-69-generic-x86_64-with-glibc2.29
Is CUDA available: True
CUDA runtime version: 11.1.105
CUDA_MODULE_LOADING set to: LAZY
GPU models and configuration:
GPU 0: NVIDIA GeForce RTX 3080
GPU 1: NVIDIA GeForce RTX 3080

Nvidia driver version: 470.182.03
cuDNN version: Could not collect
HIP runtime version: N/A
MIOpen runtime version: N/A
Is XNNPACK available: True

CPU:
Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Byte Order:                      Little Endian
Address sizes:                   43 bits physical, 48 bits virtual
CPU(s):                          64
On-line CPU(s) list:             0-63
Thread(s) per core:              2
Core(s) per socket:              32
Socket(s):                       1
NUMA node(s):                    1
Vendor ID:                       AuthenticAMD
CPU family:                      23
Model:                           49
Model name:                      AMD Ryzen Threadripper 3970X 32-Core Proces
sor
Stepping:                        0
Frequency boost:                 enabled
CPU MHz:                         2200.000
CPU max MHz:                     3700.0000
CPU min MHz:                     2200.0000
BogoMIPS:                        7400.02
Virtualization:                  AMD-V
L1d cache:                       1 MiB
L1i cache:                       1 MiB
L2 cache:                        16 MiB
L3 cache:                        128 MiB
NUMA node0 CPU(s):               0-63
Vulnerability Itlb multihit:     Not affected
```

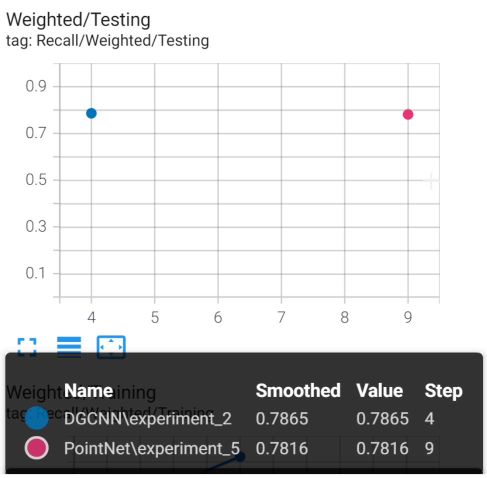
```
Vulnerability L1tf:              Not affected
Vulnerability Mds:               Not affected
Vulnerability Meltdown:          Not affected
Vulnerability Mmio stale data:   Not affected
Vulnerability Retbleed:          Mitigation; untrained return thunk; SMT ena
bled with STIBP protection
Vulnerability Spec store bypass: Mitigation; Speculative Store Bypass disabl
ed via prctl and seccomp
Vulnerability Spectre v1:        Mitigation; usercopy/swapgs barriers and __
user pointer sanitization
Vulnerability Spectre v2:        Mitigation; Retpolines, IBPB conditional, S
TIBP always-on, RSB filling, PBRSB-eIBRS Not affected
Vulnerability Srbds:             Not affected
Vulnerability Tsx async abort:   Not affected
Flags:                           fpu vme de pse tsc msr pae mce cx8 apic sep
mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ht syscall nx mmxext f
xsr_opt pdpe1gb rdtscp lm constant_tsc rep_good nopl nonstop_tsc cpuid extd_
apicid aperfmperf rapl pni pclmulqdq monitor ssse3 fma cx16 sse4_1 sse4_2 mo
vbe popcnt aes xsave avx f16c rdrand lahf_lm cmp_legacy svm extapic cr8_lega
cy abm sse4a misalignsse 3dnowprefetch osvw ibs skinit wdt tce topoext perfc
tr_core perfctr_nb bpext perfctr_llc mwaitx cpb cat_l3 cdp_l3 hw_pstate ssbd
mba ibpb stibp vmmcall fsgsbase bmi1 avx2 smep bmi2 cqm rdt_a rdseed adx sma
p clflushopt clwb sha_ni xsaveopt xsavec xgetbv1 xsaves cqm_llc cqm_occup_ll
c cqm_mbm_total cqm_mbm_local clzero irperf xsaveerptr rdpru wbnoinvd arat n
pt lbrv svm_lock nrip_save tsc_scale vmcb_clean flushbyasid decodeassists pa
usefilter pfthreshold avic v_vmsave_vmload vgif v_spec_ctrl umip rdpid overf
low_recov succor smca sme sev sev_es

Versions of relevant libraries:
[pip3] numpy==1.18.5
[pip3] torch==2.0.0
[pip3] torchsummary==1.5.1
[pip3] torchvision==0.15.1
[pip3] torchviz==0.0.2
[conda] blas                      1.0                         mkl
[conda] mkl                       2021.4.0          h06a4308_640
[conda] mkl-service               2.4.0          py310h7f8727e_0
[conda] mkl_fft                   1.3.1          py310hd6ae3a3_0
[conda] mkl_random                1.2.2          py310h00e6091_0
[conda] numpy                     1.23.5         py310hd5efca6_0
[conda] numpy-base                1.23.5         py310h8e6c178_0
Working on cuda:0
SimpleDGCNN(
  (shared): Sequential(
    (0): DynEdgeConv(
      (shared): Sequential(
        (0): Sequential(
          (0): Conv2d(6, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
          (2): ReLU(inplace=True)
        )
      )
    )
    (1): DynEdgeConv(
      (shared): Sequential(
```

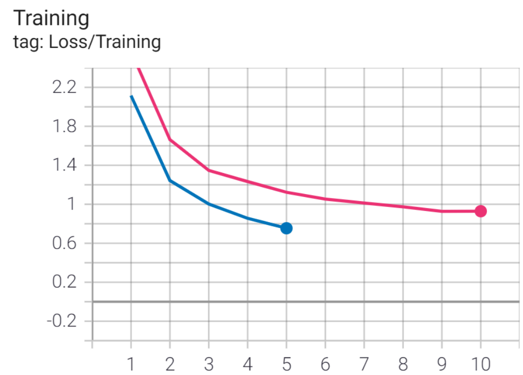
```
        (0): Sequential(
          (0): Conv2d(128, 128, kernel_size=(1, 1), stride=(1, 1), bias=Fals
e)
          (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
          (2): ReLU(inplace=True)
        )
      )
    )
    (2): DynEdgeConv(
      (shared): Sequential(
        (0): Sequential(
          (0): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fal
se)
          (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track
_running_stats=True)
          (2): ReLU(inplace=True)
        )
      )
    )
  )
  (pool): Sequential(
    (0): MaxPool1d(kernel_size=1024, stride=1024, padding=0, dilation=1, cei
l_mode=False)
    (1): Flatten(start_dim=1, end_dim=-1)
  )
  (classifier): Sequential(
    (0): Sequential(
      (0): Sequential(
        (0): Linear(in_features=1024, out_features=512, bias=False)
        (1): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
        (2): ReLU(inplace=True)
      )
    )
    (1): Sequential(
      (0): Sequential(
        (0): Linear(in_features=512, out_features=256, bias=True)
        (1): ReLU(inplace=True)
        (2): Dropout(p=0.7, inplace=False)
      )
    )
    (2): Linear(in_features=256, out_features=40, bias=True)
  )
)
Number of available threads: 32
Training from epoch 1 to 10
Epoch: 1 (TRAINING)   #################################
  0%|          | 0/1230 [00:00<?, ?it/s]
loss = 3.22737 (averaged over 9840 samples)
              recall precision  f1_score       iou
      mean     6.93%     4.96%     4.86%     2.77%
  weighted    17.10%     9.16%    10.73%     6.29%
Epoch: 1 (VALIDATION) #################################
  0%|          | 0/309 [00:00<?, ?it/s]
```

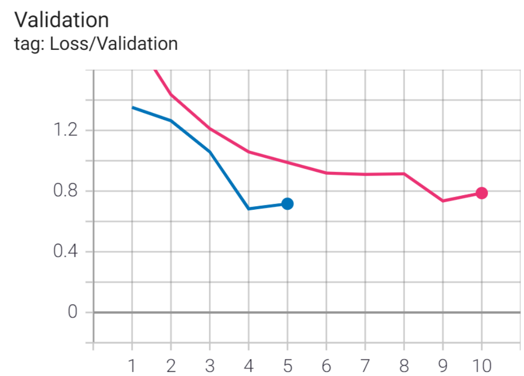
```
loss = 2.87213 (averaged over 2468 samples)
              recall precision  f1_score       iou
      mean    13.11%    12.64%     9.22%     5.81%
  weighted    20.75%    19.82%    14.39%     9.10%
Epoch: 2 (TRAINING)   #################################
  0%|          | 0/1230 [00:00<?, ?it/s]
loss = 2.79583 (averaged over 9840 samples)
              recall precision  f1_score       iou
      mean    12.76%    11.99%    10.57%     6.35%
  weighted    25.37%    18.47%    19.47%    12.25%
Epoch: 2 (VALIDATION) #################################
  0%|          | 0/309 [00:00<?, ?it/s]
loss = 2.37752 (averaged over 2468 samples)
              recall precision  f1_score       iou
      mean    20.87%    21.26%    16.82%    11.57%
  weighted    31.24%    32.24%    24.96%    17.15%
Epoch: 3 (TRAINING)   #################################
  0%|          | 0/1230 [00:00<?, ?it/s]
loss = 2.52190 (averaged over 9840 samples)
              recall precision  f1_score       iou
      mean    16.98%    19.00%    15.55%     9.57%
  weighted    30.23%    25.24%    25.58%    16.66%
Epoch: 3 (VALIDATION) #################################
  0%|          | 0/309 [00:00<?, ?it/s]
loss = 1.96482 (averaged over 2468 samples)
              recall precision  f1_score       iou
      mean    26.81%    28.71%    21.71%    15.99%
  weighted    39.06%    35.80%    31.42%    23.59%
Epoch: 4 (TRAINING)   #################################
  0%|          | 0/1230 [00:00<?, ?it/s]
loss = 2.36709 (averaged over 9840 samples)
              recall precision  f1_score       iou
      mean    20.13%    21.25%    18.53%    11.72%
  weighted    33.79%    28.50%    29.31%    19.55%
Epoch: 4 (VALIDATION) #################################
  0%|          | 0/309 [00:00<?, ?it/s]
loss = 1.80998 (averaged over 2468 samples)
              recall precision  f1_score       iou
      mean    34.24%    38.12%    30.98%    22.88%
  weighted    46.88%    47.96%    41.67%    31.31%
Epoch: 5 (TRAINING)   #################################
  0%|          | 0/1230 [00:00<?, ?it/s]
loss = 2.25857 (averaged over 9840 samples)
              recall precision  f1_score       iou
      mean    22.41%    26.54%    21.32%    13.66%
  weighted    36.44%    33.04%    32.67%    22.19%
Epoch: 5 (VALIDATION) #################################
  0%|          | 0/309 [00:00<?, ?it/s]
loss = 1.73468 (averaged over 2468 samples)
              recall precision  f1_score       iou
      mean    35.30%    39.19%    31.39%    23.00%
  weighted    48.14%    47.41%    41.88%    30.79%
Epoch: 6 (TRAINING)   #################################
  0%|          | 0/1230 [00:00<?, ?it/s]
```

```
loss = 2.18511 (averaged over 9840 samples)
              recall precision  f1_score       iou
      mean    23.77%    26.19%    22.80%    14.76%
  weighted    38.30%    33.93%    34.52%    23.77%
Epoch: 6 (VALIDATION) #################################
  0%|          | 0/309 [00:00<?, ?it/s]
loss = 1.62741 (averaged over 2468 samples)
              recall precision  f1_score       iou
      mean    37.93%    42.53%    34.54%    24.83%
  weighted    51.13%    54.67%    47.06%    34.66%
Epoch: 7 (TRAINING)   #################################
  0%|          | 0/1230 [00:00<?, ?it/s]
loss = 2.09778 (averaged over 9840 samples)
              recall precision  f1_score       iou
      mean    25.05%    27.82%    24.21%    15.80%
  weighted    40.07%    35.97%    36.28%    25.21%
Epoch: 7 (VALIDATION) #################################
  0%|          | 0/309 [00:00<?, ?it/s]
loss = 1.53715 (averaged over 2468 samples)
              recall precision  f1_score       iou
      mean    41.41%    49.20%    37.77%    28.23%
  weighted    53.40%    54.89%    47.50%    36.24%
Epoch: 8 (TRAINING)   #################################
  0%|          | 0/1230 [00:00<?, ?it/s]
loss = 2.08281 (averaged over 9840 samples)
              recall precision  f1_score       iou
      mean    25.98%    29.15%    25.35%    16.54%
  weighted    41.18%    37.63%    37.69%    26.23%
Epoch: 8 (VALIDATION) #################################
  0%|          | 0/309 [00:00<?, ?it/s]
loss = 1.56135 (averaged over 2468 samples)
              recall precision  f1_score       iou
      mean    40.09%    43.22%    36.43%    26.86%
  weighted    52.88%    55.83%    48.05%    36.25%
Epoch: 9 (TRAINING)   #################################
  0%|          | 0/1230 [00:00<?, ?it/s]
loss = 1.99788 (averaged over 9840 samples)
              recall precision  f1_score       iou
      mean    28.25%    31.49%    27.60%    18.29%
  weighted    43.43%    40.03%    39.96%    28.25%
Epoch: 9 (VALIDATION) #################################
  0%|          | 0/309 [00:00<?, ?it/s]
loss = 1.46631 (averaged over 2468 samples)
              recall precision  f1_score       iou
      mean    43.89%    51.57%    41.30%    30.43%
  weighted    55.92%    58.01%    50.39%    37.99%
Epoch: 10 (TRAINING)  #################################
  0%|          | 0/1230 [00:00<?, ?it/s]
loss = 1.93540 (averaged over 9840 samples)
              recall precision  f1_score       iou
      mean    28.64%    31.56%    28.13%    18.74%
  weighted    44.09%    40.49%    40.70%    29.04%
Epoch: 10 (VALIDATION)#################################
  0%|          | 0/309 [00:00<?, ?it/s]
```

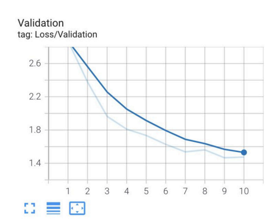
```
loss = 1.47490 (averaged over 2468 samples)
               recall precision  f1_score       iou
      mean     45.03%     46.54%     40.86%     30.25%
  weighted     54.66%     56.56%     50.15%     38.00%
Epoch: 10 (TESTING)   #################################
  0%|          | 0/309 [00:00<?, ?it/s]
loss = 1.47490 (averaged over 2468 samples)
               recall precision  f1_score       iou
      mean     45.03%     46.54%     40.86%     30.25%
  weighted     54.66%     56.56%     50.15%     38.00%
```

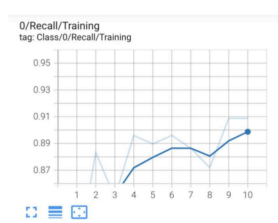


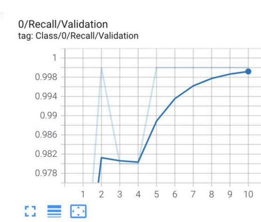




TODO: Attach a screenshot of the Loss and Recall tabs from TensorBoard in a text cell below:

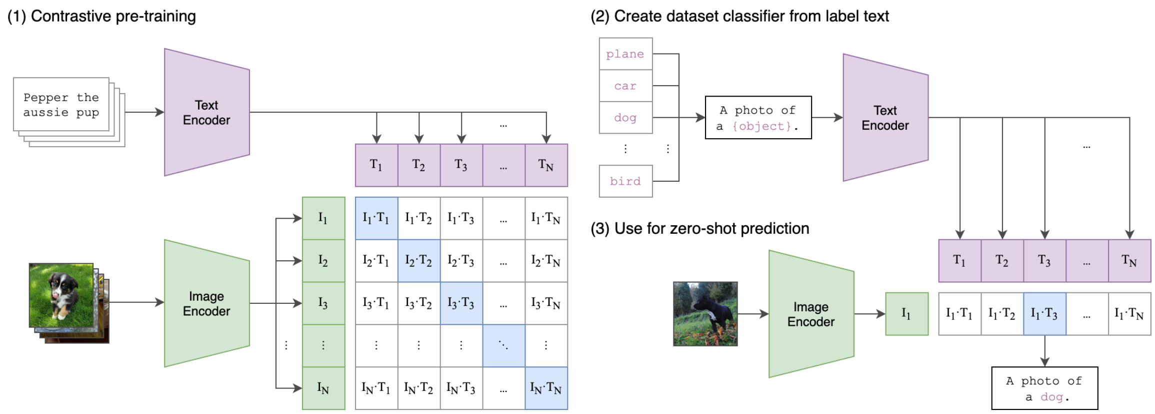






Part 4: Zero-shot Point Cloud Classification using CLIP (1.5 points)

CLIP (Contrastive Language-Image Pre-Training) is a neural network trained on a variety of (image, text) pairs. It can be instructed in natural language to predict the most relevant text snippet, given an image, without directly optimizing for the task, similarly to the zero-shot capabilities of GPT-2 and 3. CLIP is able to match the performance of the original ResNet50 on ImageNet “zero-shot” without using any of the original 1.28M labeled examples, overcoming several major challenges in computer vision.



Task 1: A Toy Version of PointCLIP (1.5 Point)

We are going to ultize the zero-shot ability of CLIP for point cloud classification without any labelling. The idea of using CLIP for point cloud classificaiton was firstly proposed in PointCLIP paper in Dec 2021. We encourge the students to read the CLIP blog, the colab demo of CLIP and the most relevent PointCLIP paper before moving on to the following tasks.

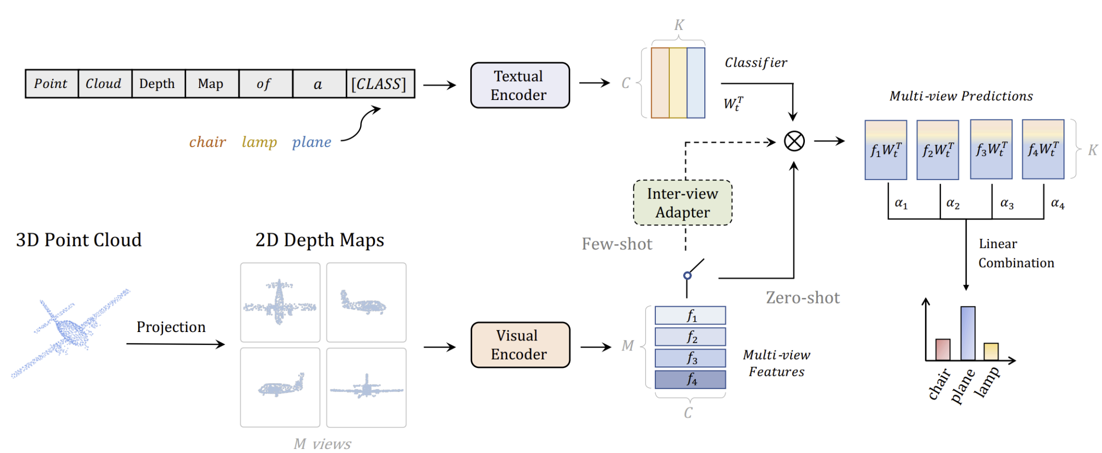


Figure 2. **The Pipeline of PointCLIP.** To bridge the modal gap, PointCLIP projects the point cloud onto multi-view depth maps, and conducts 3D recognition via CLIP pre-trained in 2D. The switch provides alternatives for direct zero-shot classification and few-shot classification with inter-view adapter, respectively, in solid and dotted lines.

As a part in a course project, we do not require to implement the whole framework of PointCLIP. Instead, we just work on a single view version without any training on the target dataset. Further, we do not require testing PointCLIP on all pointclouds, we just try it out on one simple sample, a random sample from the airplane class. Even more, we do not require the students to write a rendering function to render an image from the point cloud as done in MVTN, instead I provide a simple screenshot of the point cloud.

In [1]:
```
import numpy as np
import torch
from pkg_resources import packaging
from PIL import Image, ImageOps
import clip
```

In [2]:
```
device = "cuda" if torch.cuda.is_available() else "cpu"
model, preprocess = clip.load("ViT-B/32", device=device)
```

```
/home/ubuntu/.local/lib/python3.8/site-packages/clip/clip.py:57: UserWarnin
g: /home/ubuntu/.cache/clip/ViT-B-32.pt exists, but the SHA256 checksum does
not match; re-downloading the file
  warnings.warn(f"{download_target} exists, but the SHA256 checksum does not
match; re-downloading the file")
100%|████████████████████████████████████████| 338M/338M [00:21<00:00, 16.7Mi
B/s]
```

In [3]:
```
# download image
!gdown https://drive.google.com/uc?id=1m9ychgioKZqrUanxQpp8kbYgQcFqEoH5
```

```
Downloading...
From: https://drive.google.com/uc?id=1m9ychgioKZqrUanxQpp8kbYgQcFqEoH5
To: /home/ubuntu/Documents/DeepLearning/Repos_DL/airplane.png
100%|██████████████████████████████████████████| 106k/106k [00:00<00:00, 413k
B/s]
```

In [21]:
```
# open a random view of point cloud
im = Image.open("airplane.png")
image = preprocess(im).unsqueeze(0).cuda() # pre-process images (check CLIP
```

In [33]:
```
# TODO: CLIP for zero-shot point cloud classification (1.5 points)

text_descriptions = [f"Point Cloud depth map of a {obj}" for obj in test_set
# text_descriptions = [f"Point Cloud depth map of a {obj}" for obj in ["plan
print(text_descriptions)
text_tokens = clip.tokenize(text_descriptions).cuda()


with torch.no_grad():
    image_features = model.encode_image(image).float()
    text_features = model.encode_text(text_tokens).float()
    text_features /= text_features.norm(dim=-1, keepdim=True)

text_probs = (100 * image_features @ text_features.T).softmax(dim=-1)
top_probs, top_labels = text_probs.cpu().topk(5, dim=-1)
top_probs = top_probs.numpy()
top_labels = top_labels.numpy()
```

```
['Point Cloud depth map of a airplane', 'Point Cloud depth map of a bathtu
b', 'Point Cloud depth map of a bed', 'Point Cloud depth map of a bench', 'P
oint Cloud depth map of a bookshelf', 'Point Cloud depth map of a bottle',
'Point Cloud depth map of a bowl', 'Point Cloud depth map of a car', 'Point
Cloud depth map of a chair', 'Point Cloud depth map of a cone', 'Point Cloud
depth map of a cup', 'Point Cloud depth map of a curtain', 'Point Cloud dept
h map of a desk', 'Point Cloud depth map of a door', 'Point Cloud depth map
of a dresser', 'Point Cloud depth map of a flower_pot', 'Point Cloud depth m
ap of a glass_box', 'Point Cloud depth map of a guitar', 'Point Cloud depth
map of a keyboard', 'Point Cloud depth map of a lamp', 'Point Cloud depth ma
p of a laptop', 'Point Cloud depth map of a mantel', 'Point Cloud depth map
of a monitor', 'Point Cloud depth map of a night_stand', 'Point Cloud depth
map of a person', 'Point Cloud depth map of a piano', 'Point Cloud depth map
of a plant', 'Point Cloud depth map of a radio', 'Point Cloud depth map of a
range_hood', 'Point Cloud depth map of a sink', 'Point Cloud depth map of a
sofa', 'Point Cloud depth map of a stairs', 'Point Cloud depth map of a stoo
l', 'Point Cloud depth map of a table', 'Point Cloud depth map of a tent',
'Point Cloud depth map of a toilet', 'Point Cloud depth map of a tv_stand',
'Point Cloud depth map of a vase', 'Point Cloud depth map of a wardrobe', 'P
oint Cloud depth map of a xbox']
```

In [34]:

```python
# visulize results
import matplotlib.pyplot as plt

plt.figure(figsize=(16, 16))

plt.subplot(1, 2, 1)
plt.imshow(im)
plt.axis("off")

plt.subplot(1, 2, 2)
y = np.arange(top_probs.shape[-1])
plt.grid()
print(y)
print(top_probs[0])
plt.barh(y, top_probs[0])
plt.gca().invert_yaxis()
plt.gca().set_axisbelow(True)
plt.yticks(y, [test_set.classes[index] for index in top_labels[0]])
plt.xlabel("probability")

plt.subplots_adjust(wspace=0.5)
plt.show()
```

```
[0 1 2 3 4]
[1.0000000e+00 7.5598556e-26 4.4544227e-28 3.0520654e-29 2.0386435e-29]
```

TODO: Congrats on fininshing the coding part. For the purpose of assesement, please do not delete the output from each cell. Also, the students should prepare for the questions related to PointNet, message passing, DGCNN, CLIP, and Point-CLIP.

Concluding Remark

I hope these projects gave you a hands-on experience with deep learning. Now is definitely an exciting time for AI. We barely scratched the surface but the posibilities are wide open. If you want to see what else is out there, maybe start with reinforcement learning and self-supervised learning.