به نام خدا

مبانی بینایی کامپیوتر

دکتر هانیه نادری

گزارش پروژه PointConv

کیارش کیانیان - ۹۹۱۰۵۶۷۸

امیررضا ابوطالبی - ۹۹۱۰۵۱۹۷

مهرشاد برزمینی - ۹۹۱۷۰۳۶۱

بهار ۱۴۰۳

مقدمه

مقاله "PointConv: شبکه‌های عصبی کانولوشن عمیق بر روی ابرنقاط (point clouds) سه‌بعدی یک روش جدید برای اعمال عملیات کانولوشن بر روی ابرنقاط سه‌بعدی ارائه می‌دهد که به دلیل ماهیت نامنظم و بدون ترتیب این داده‌ها، روش‌های کانولوشن سنتی برای شبکه‌های عصبی کانولوشن دوبعدی (CNN) برای آن‌ها قابل استفاده نیستند. نویسندگان یک عملیات کانولوشن جدید به نام PointConv پیشنهاد می‌کنند که می‌تواند برای ساخت شبکه‌های کانولوشن عمیق برای داده‌های ابرنقاط سه‌بعدی استفاده شود. این روش به خصوص در برنامه‌های اخیر مانند رباتیک، رانندگی خودکار و واقعیت مجازی/افزوده که حسگرها می‌توانند داده‌های سه‌بعدی را به طور مستقیم دریافت کنند، بسیار مهم است.

بیان مسئله

در تصاویر دوبعدی، شبکه‌های عصبی کانولوشن (CNNs) با استفاده از ویژگی ترجمه‌ناپذیری، انقلابی در وظایف بینایی کامپیوتر ایجاد کرده‌اند. با این حال، اعمال CNNها بر روی ابرنقاط سه‌بعدی به دلیل ماهیت نامنظم و بدون ترتیب این داده‌ها چالش‌برانگیز است. روش‌های سنتی که داده‌های سه‌بعدی را به تصاویر دوبعدی یا شبکه‌های حجمی سه‌بعدی تبدیل می‌کنند، محدودیت‌هایی در وضوح و کارایی محاسباتی دارند.

رویکرد PointConv

PointConv فیلتر دینامیکی را به یک عملیات کانولوشن جدید برای ابرنقاط گسترش می‌دهد. هسته‌های کانولوشن در PointConv به عنوان توابع غیرخطی از مختصات محلی نقاط سه‌بعدی در نظر گرفته می‌شوند که شامل توابع وزن و چگالی هستند. این توابع با استفاده از شبکه‌های پرسپترون چندلایه (MLPs) و تخمین چگالی هسته‌ای یاد گرفته می‌شوند. مشارکت کلیدی PointConv یک فرمول‌بندی کارآمد برای محاسبه توابع وزن است که امکان افزایش مقیاس شبکه‌ها و بهبود عملکرد قابل توجه را فراهم می‌کند.

مشارکت‌های کلیدی

1. **عملیات PointConv**: PointConv تقریب کانولوشن پیوسته سه‌بعدی بر روی مجموعه نقاط را ارائه می‌دهد که ترجمه‌ناپذیر و بدون ترتیب است.

2. **بهره‌وری حافظه**: یک رویکرد فرموله شده مجدد برای اجرای PointConv که به طور قابل توجهی مصرف حافظه را کاهش می‌دهد و امکان افزایش مقیاس به سطح شبکه‌های CNN مدرن را فراهم می‌کند.
3. **PointDeconv**: گسترش PointConv به دکانولوشن که برای وظایفی مانند تقسیم‌بندی که نیاز به انتقال اطلاعات از لایه‌های درشت به لایه‌های دقیق دارد، ضروری است.

این بخش‌ها مقدمه‌ای برای فهم بهتر مقاله و روش‌های ارائه شده در آن فراهم می‌کنند. در ادامه به توضیحات بیشتری درباره کدهای پیاده‌سازی شده و نحوه عملکرد آن‌ها خواهیم پرداخت.مدلی که ما برای تست کردن روش گفته شده انتخاب کردیم که در دسترس بود، مدل ModelNet40 stanford بود و بر اساس آن train داده ها انجام شده است.

تحلیل کد

حال به تحلیل کد ها میپردازیم. تا جای ممکن سعی کردیم توابع کلیدی و بخش های مهم پروژه را در این گزارش توضیح دهیم. :

ابتدا به فایل eval_cls_conv.py میپردازیم:

بخش زیر پارامترهای ورودی را با استفاده از `parseargs` تعریف می‌کند که شامل اندازه بچ، دستگاه GPU، مسیر نقطه بازیابی (checkpoint)، تعداد نقاط و سایر پارامترها می‌باشد.

```
def parse_args():
    '''PARAMETERS'''
    parser = argparse.ArgumentParser('PointConv')
    parser.add_argument('--batchsize', type=int, default=32, help='batch size')
    parser.add_argument('--gpu', type=str, default='0', help='specify gpu device')
    parser.add_argument('--checkpoint', type=str, default=None, help='checkpoint')
    parser.add_argument('--num_point', type=int, default=1024, help='Point Number [default: 1024]')
    parser.add_argument('--num_workers', type=int, default=16, help='Worker Number [default: 16]')
    parser.add_argument('--model_name', default='pointconv', help='model name')
    parser.add_argument('--normal', action='store_true', default=False, help='Whether to use normal information [default: False]')
    return parser.parse_args()
```

پس از آن تابع main را داریم که ارزیابی اصلی را انجام میدهد که شامل موارد زیر است:
تنظیمات اولیه: در این بخش محیط برای استفاده از GPU تنظیم می‌شود و دایرکتوری‌های لازم برای ذخیره نتایج و لاگ‌ها ایجاد می‌شوند.
بارگذاری داده‌ها: داده‌های تست از دیتاست ModelNet40 بارگذاری می‌شوند.

بارگذاری مدل: مدل PointConv بارگذاری شده و اگر نقطه بازیابی موجود باشد، مدل از آن نقطه بارگذاری می‌شود.

ارزیابی مدل: مدل بر روی داده‌های تست ارزیابی شده و دقت کل محاسبه و نمایش داده می‌شود.

```
'''CREATE DIR'''
experiment_dir = Path('./eval_experiment/')
experiment_dir.mkdir(exist_ok=True)
file_dir = Path(str(experiment_dir) + '/%s_ModelNet40-'%args.model_name + str(datetime.datetime.now().strftime('%Y-%m-%d_%H-%M')))
file_dir.mkdir(exist_ok=True)
checkpoints_dir = file_dir.joinpath('checkpoints/')
checkpoints_dir.mkdir(exist_ok=True)
os.system('cp %s %s' % (args.checkpoint, checkpoints_dir))
log_dir = file_dir.joinpath('logs/')
log_dir.mkdir(exist_ok=True)

'''LOG'''
args = parse_args()
logger = logging.getLogger(args.model_name)
logger.setLevel(logging.INFO)
formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
file_handler = logging.FileHandler(str(log_dir) + 'eval_%s_cls.txt'%args.model_name)
file_handler.setLevel(logging.INFO)
file_handler.setFormatter(formatter)
logger.addHandler(file_handler)
logger.info('---------------------------------------------------EVAL---------------------------------------------------')
logger.info('PARAMETER ...')
logger.info(args)
```

```
'''DATA LOADING'''
logger.info('Load dataset ...')
DATA_PATH = './data/modelnet40_normal_resampled/'

TEST_DATASET = ModelNetDataLoader(root=DATA_PATH, npoint=args.num_point, split='test', normal_channel=args.normal)
testDataLoader = torch.utils.data.DataLoader(TEST_DATASET, batch_size=args.batchsize, shuffle=False, num_workers=args.num_workers)
logger.info( msg: "The number of test data is: %d",  *args: len(TEST_DATASET))

seed = 3
torch.manual_seed(seed)
if torch.cuda.is_available():
    torch.cuda.manual_seed_all(seed)

'''MODEL LOADING'''
num_class = 40
classifier = PointConvClsSsg(num_class).cuda()
if args.checkpoint is not None:
    print('Load CheckPoint...')
    logger.info('Load CheckPoint')
    checkpoint = torch.load(args.checkpoint)
    start_epoch = checkpoint['epoch']
    classifier.load_state_dict(checkpoint['model_state_dict'])
else:
    print('Please load Checkpoint to eval...')
    sys.exit(0)
```

```
'''EVAL'''
logger.info('Start evaluating...')
print('Start evaluating...')

classifier = classifier.eval()
mean_correct = []
for batch_id, data in tqdm(enumerate(testDataLoader, 0), total=len(testDataLoader), smoothing=0.9):
    pointcloud, target = data
    target = target[:, 0]

    points = pointcloud.permute(0, 2, 1)
    points, target = points.cuda(), target.cuda()
    with torch.no_grad():
        pred = classifier(points[:, :3, :], points[:, 3:, :])
    pred_choice = pred.data.max(1)[1]
    correct = pred_choice.eq(target.long().data).cpu().sum()

    mean_correct.append(correct.item()/float(points.size()[0]))

accuracy = np.mean(mean_correct)
print('Total Accuracy: %f'%accuracy)

logger.info('Total Accuracy: %f'%accuracy)
logger.info('End of evaluation...')
```

حال به کد provider.py میپردازیم:

این فایل کد شامل توابعی برای پیش‌پردازش و پردازش داده‌های ابرنقاط است.

تابع `normalize_data`: این تابع داده‌ها را نرمال‌سازی می‌کند به طوری که مختصات نقاط حول مرکز قرار گیرند و مقیاس‌بندی شوند.

```python
def normalize_data(batch_data):
    """ Normalize the batch data, use coordinates of the block centered at origin,
        Input:
            BxNxC array
        Output:
            BxNxC array
    """
    B, N, C = batch_data.shape
    normal_data = np.zeros((B, N, C))
    for b in range(B):
        pc = batch_data[b]
        centroid = np.mean(pc, axis=0)
        pc = pc - centroid
        m = np.max(np.sqrt(np.sum(pc ** 2, axis=1)))
        pc = pc / m
        normal_data[b] = pc
    return normal_data
```

تابع `shuffle_data`: داده‌ها و برچسب‌ها را به صورت تصادفی مرتب می‌کند.

```python
def shuffle_data(data, labels):
    """ Shuffle data and labels.
        Input:
          data: B,N,... numpy array
          label: B,... numpy array
        Return:
          shuffled data, label and shuffle indices
    """
    idx = np.arange(len(labels))
    np.random.shuffle(idx)
    return data[idx, ...], labels[idx], idx
```

تابع `rotate_point_cloud`: ابرنقاط را به صورت تصادفی حول محور های مختلف میچرخاند که چند مدل از آن را داریم که در پایین به یکی از آنها اشاره کردیم.

```python
def rotate_point_cloud(batch_data):
    """ Randomly rotate the point clouds to augument the dataset
        rotation is per shape based along up direction
        Input:
          BxNx3 array, original batch of point clouds
        Return:
          BxNx3 array, rotated batch of point clouds
    """
    rotated_data = np.zeros(batch_data.shape, dtype=np.float32)
    for k in range(batch_data.shape[0]):
        rotation_angle = np.random.uniform() * 2 * np.pi
        cosval = np.cos(rotation_angle)
        sinval = np.sin(rotation_angle)
        rotation_matrix = np.array([[cosval, 0, sinval],
                                    [0, 1, 0],
                                    [-sinval, 0, cosval]])
        shape_pc = batch_data[k, ...]
        rotated_data[k, ...] = np.dot(shape_pc.reshape((-1, 3)), rotation_matrix)
    return rotated_data
```

تابع `jitter_point_cloud`: نقاط را به صورت تصادفی جابجا می‌کند تا تنوع بیشتری به داده‌ها اضافه کند.

```python
def jitter_point_cloud(batch_data, sigma=0.01, clip=0.05):
    """ Randomly jitter points. jittering is per point.
        Input:
          BxNx3 array, original batch of point clouds
        Return:
          BxNx3 array, jittered batch of point clouds
    """
    B, N, C = batch_data.shape
    assert(clip > 0)
    jittered_data = np.clip(sigma * np.random.randn(B, N, C), -1*clip, clip)
    jittered_data += batch_data
    return jittered_data
```

بقیه توابع مربوطه در فایل provider.py نیز داریم که به همین منظور های گفته شده نوشته شده اند که توضیحات لازم در کد آنها کامنت شده است.

توضیحات مربوط به فایل train_cls_conv.py: این فایل کد برای آموزش مدل PointConv بر روی دیتاست ModelNet40 استفاده می‌شود. کد شامل تنظیمات اولیه، بارگذاری داده‌ها، تعریف مدل، و اجرای فرآیند آموزش است. در اینجا به تفصیل به هر یک از بخش‌ها و توابع اصلی پرداخته می‌شود.

در ابتدا توسط تابع parse_args پارامترهای ورودی را تعریف و تجزیه می‌کند. پارامترهایی مانند اندازه بچ، تعداد ایپوک‌ها، نرخ یادگیری و دستگاه GPU از جمله مواردی هستند که کاربر می‌تواند آنها را تنظیم کند.

در تابع main فرایند زیر را داریم:

در این بخش، دستگاه GPU مشخص می‌شود و دایرکتوری‌های لازم برای ذخیره نتایج و نقاط بازیابی (checkpoints) ایجاد می‌شوند. سپس برای ثبت لاگ‌های فرآیند آموزش استفاده می‌شود. اطلاعات مربوط به پارامترهای ورودی و وضعیت‌های مختلف فرآیند آموزش در فایل لاگ ثبت می‌شوند.

```
1 usage   Dylan Wu +1
def main(args):
    '''HYPER PARAMETER'''
    os.environ["CUDA_VISIBLE_DEVICES"] = args.gpu

    '''CREATE DIR'''
    experiment_dir = Path('/kaggle/working/experiment/')
    experiment_dir.mkdir(exist_ok=True)
    file_dir = Path(str(experiment_dir) + '/%s_ModelNet40-' % args.model_name + str(
        datetime.datetime.now().strftime('%Y-%m-%d_%H-%M')))
    file_dir.mkdir(exist_ok=True)
    checkpoints_dir = file_dir.joinpath('checkpoints/')
    checkpoints_dir.mkdir(exist_ok=True)
    log_dir = file_dir.joinpath('logs/')
    log_dir.mkdir(exist_ok=True)

    '''LOG'''
    args = parse_args()
    logger = logging.getLogger(args.model_name)
    logger.setLevel(logging.INFO)
    formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
    file_handler = logging.FileHandler(str(log_dir) + 'train_%s_cls.txt' % args.model_name)
    file_handler.setLevel(logging.INFO)
    file_handler.setFormatter(formatter)
    logger.addHandler(file_handler)
    logger.info(
        '----------------------------------------------------TRANING----------------------------------------------------')
    logger.info('PARAMETER ...')
    logger.info(args)
```

و بعد بارگذاری داده ها را داریم. در این بخش، داده‌های آموزشی و تست از دیتاست ModelNet40 بارگذاری می‌شوند. از کلاس `ModelNetDataLoader` برای این منظور استفاده شده است که داده‌ها را با تعداد نقاط مشخص بارگذاری می‌کند.

```
'''DATA LOADING'''
logger.info('Load dataset ...')
print(os.getcwd())

DATA_PATH = '/kaggle/input/modelnet40-princeton-3d-object-dataset/ModelNet40'  # Update this to your dataset path

TRAIN_DATASET = ModelNetDataLoader(root=DATA_PATH, npoint=args.num_point, split='train', normal_channel=args.normal)
TEST_DATASET = ModelNetDataLoader(root=DATA_PATH, npoint=args.num_point, split='test', normal_channel=args.normal)
trainDataLoader = torch.utils.data.DataLoader(TRAIN_DATASET, batch_size=args.batchsize, shuffle=True,
                                              num_workers=args.num_workers, collate_fn=custom_collate_fn)
testDataLoader = torch.utils.data.DataLoader(TEST_DATASET, batch_size=args.batchsize, shuffle=False,
                                             num_workers=args.num_workers)

logger.info( msg: "The number of training data is: %d",  *args: len(TRAIN_DATASET))
logger.info( msg: "The number of test data is: %d",  *args: len(TEST_DATASET))
```

سپس برای اطمینان از تکرارپذیری نتایج، seed تصادفی برای PyTorch تنظیم می‌شود.

بعد بارگزاری مدل را داریم. در این بخش، مدل PointConv بارگذاری می‌شود. اگر مدل پیش‌آموزش موجود باشد، از آن استفاده می‌شود و اگر نباشد، آموزش از ابتدا شروع می‌شود.

```
'''MODEL LOADING'''
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
num_class = 40
classifier = PointConvClsSsg(num_class).to(device)
if args.pretrain is not None:
    print('Use pretrain model...')
    logger.info('Use pretrain model')
    checkpoint = torch.load(args.pretrain, map_location=device)
    start_epoch = checkpoint['epoch']
    classifier.load_state_dict(checkpoint['model_state_dict'])
else:
    print('No existing model, starting training from scratch...')
    start_epoch = 0
```

سپس یک optimizer تنظیم میکنیم. بهینه‌ساز مدل بسته به پارامتر ورودی می‌تواند `SGD` یا `Adam` باشد. همچنین یک زمان‌بندی‌کننده برای نرخ یادگیری تنظیم می‌شود.

```
if args.optimizer == 'SGD':
    optimizer = torch.optim.SGD(classifier.parameters(), lr=0.01, momentum=0.9)
elif args.optimizer == 'Adam':
    optimizer = torch.optim.Adam(
        classifier.parameters(),
        lr=args.learning_rate,
        betas=(0.9, 0.999),
        eps=1e-08,
        weight_decay=args.decay_rate
    )
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=30, gamma=0.7)
global_epoch = 0
global_step = 0
best_tst_accuracy = 0.0
blue = lambda x: '\033[94m' + x + '\033[0m'
```

سپس فرایند آموزش را داریم که به توضیحات آن میپردازیم:
شروع آموزش: آموزش مدل برای تعداد ایپوک‌های مشخص شده آغاز می‌شود. هر ایپوک شامل چندین بچ است که هر بچ به مدل داده می‌شود.

پیش‌پردازش داده‌ها: داده‌ها با استفاده از توابع موجود در `provider` برای افزایش داده‌ها (augmentation) پردازش می‌شوند.

آموزش مدل: مدل با داده‌های پردازش شده آموزش داده می‌شود. خروجی مدل و برچسب‌های واقعی با استفاده از تابع `nll_loss` برای محاسبه خطا استفاده می‌شوند. سپس بهینه‌ساز برای به‌روزرسانی وزن‌های مدل استفاده می‌شود.

ارزیابی و ذخیره مدل: مدل بر روی داده‌های تست ارزیابی می‌شود و دقت محاسبه می‌شود. اگر دقت تست بهتر از بهترین دقت قبلی باشد، مدل ذخیره می‌شود.

فایل utils.py:

این فایل شامل توابع کمکی برای فرآیند آموزش و ارزیابی مدل است.

تابع `to_categorical`: این تابع برچسب‌ها را به صورت یک‌داغ (one-hot) کدگذاری می‌کند.

```
2 usages
def to_categorical(y, num_classes):
    """ 1-hot encodes a tensor """
    new_y = torch.eye(num_classes)[y.cpu().data.numpy(),]
    if (y.is_cuda):
        return new_y.cuda()
    return new_y
```

تابع `save_checkpoint`: این تابع مدل را به همراه وضعیت بهینه‌ساز در یک نقطه بازیابی (checkpoint) ذخیره می‌کند.

```
3 usages
def save_checkpoint(epoch, train_accuracy, test_accuracy, model, optimizer, path, modelnet='checkpoint'):
    savepath = path + '/%s-%f-%04d.pth' % (modelnet, test_accuracy, epoch)
    state = {
        'epoch': epoch,
        'train_accuracy': train_accuracy,
        'test_accuracy': test_accuracy,
        'model_state_dict': model.state_dict(),
        'optimizer_state_dict': optimizer.state_dict(),
    }
    torch.save(state, savepath)
```

تابع `test`: این تابع مدل را بر روی داده‌های تست ارزیابی کرده و دقت کل را محاسبه می‌کند.

```python
def test(model, loader):
    total_correct = 0.0
    total_seen = 0.0
    for j, data in enumerate(loader, 0):
        points, target = data
        target = target[:, 0]
        points = points.transpose(2, 1)
        points, target = points.cuda(), target.cuda()
        classifier = model.eval()
        with torch.no_grad():
            pred = classifier(points[:, :3, :], points[:, 3:, :])
        pred_choice = pred.data.max(1)[1]
        correct = pred_choice.eq(target.long().data).cpu().sum()
        total_correct += correct.item()
        total_seen += float(points.size()[0])

    accuracy = total_correct / total_seen
    return accuracy
```

فایل pointconv.py:

این کلاس مدل PointConv برای طبقه‌بندی را تعریف می‌کند و بعد شامل موارد زیر است:

تعریف لایه‌های مدل: مدل شامل سه لایه جذب مجموعه (Set Abstraction) و سه لایه کاملاً متصل (Fully Connected) است.

تابع `forward`: این تابع اجرای جلوروی مدل را تعریف می‌کند که ورودی نقاط و ویژگی‌ها را گرفته و خروجی دسته‌بندی را تولید می‌کند.

```python
class PointConvDensityClsSsg(nn.Module):
    def __init__(self, num_classes = 40):
        super(PointConvDensityClsSsg, self).__init__()
        feature_dim = 3
        self.sa1 = PointConvDensitySetAbstraction(npoint=512, nsample=32, in_channel=feature_dim + 3, mlp=[64, 64, 128], bandwidth = 0.1, group_all=False)
        self.sa2 = PointConvDensitySetAbstraction(npoint=128, nsample=64, in_channel=128 + 3, mlp=[128, 128, 256], bandwidth = 0.2, group_all=False)
        self.sa3 = PointConvDensitySetAbstraction(npoint=1, nsample=None, in_channel=256 + 3, mlp=[256, 512, 1024], bandwidth = 0.4, group_all=True)
        self.fc1 = nn.Linear( in_features: 1024,  out_features: 512)
        self.bn1 = nn.BatchNorm1d(512)
        self.drop1 = nn.Dropout(0.7)
        self.fc2 = nn.Linear( in_features: 512,  out_features: 256)
        self.bn2 = nn.BatchNorm1d(256)
        self.drop2 = nn.Dropout(0.7)
        self.fc3 = nn.Linear( in_features: 256, num_classes)
```

```python
def forward(self, xyz, feat):
    B, _, _ = xyz.shape
    l1_xyz, l1_points = self.sa1(xyz, feat)
    l2_xyz, l2_points = self.sa2(l1_xyz, l1_points)
    l3_xyz, l3_points = self.sa3(l2_xyz, l2_points)
    x = l3_points.view(B, 1024)
    x = self.drop1(F.relu(self.bn1(self.fc1(x))))
    x = self.drop2(F.relu(self.bn2(self.fc2(x))))
    x = self.fc3(x)
    x = F.log_softmax(x, -1)
    return x
```

حال در google Colab دیتاست را دریافت میکنیم. علت این کار وجود GPU در colab است.
دیتاست را نیز در google drive آپلود میکنیم و در colab آن را mount میکنیم.

```python
# Define the path to your file in Google Drive
drive_file_path = '/content/drive/MyDrive/datasets/modelnet40_normal_resampled.zip'

# Define the destination path in Colab
colab_destination_path = '/content/modelnet40_normal_resampled.zip'

# Copy the file from Google Drive to Colab
shutil.copy(drive_file_path, colab_destination_path)

# Optional: Unzip the file if needed
with zipfile.ZipFile(colab_destination_path, 'r') as zip_ref:
    zip_ref.extractall('/content')
```

```python
dataset_path = '/content/modelnet40_normal_resampled'
files = os.listdir(dataset_path)
print(files)

# Get the list of categories (directories)
categories = [d for d in os.listdir(dataset_path) if os.path.isdir(os.path.join(dataset_path, d))]
print(f"Total number of categories: {len(categories)}")
print("Categories:", categories)

# Count the number of files in each category
category_file_counts = {}
for category in categories:
    category_path = os.path.join(dataset_path, category)
    files = os.listdir(category_path)
    category_file_counts[category] = len(files)

# Print the number of files in each category
for category, count in category_file_counts.items():
    print(f"{category}:{count}", end=" ")
```

از توابع visualize که میبینید برای نمایش و بررسی صحت و درستی داده های دریافتی از دیتاست استفاده میکنیم. سپس به train و evaluate بر روی آنها با استفاده از کدی که پیاده سازی کردیم میپردازیم.

```
def load_txt(file_path):
    return np.loadtxt(file_path, delimiter=',')

# Function to visualize a point cloud
def visualize_point_cloud(points):
    fig = plt.figure(figsize=(15, 10))

    # Subplot 1: Only Points
    ax1 = fig.add_subplot(121, projection='3d')
    ax1.scatter(points[:, 0], points[:, 1], points[:, 2], s=1, c='k')
    ax1.set_title('Only Points')
    ax1.set_xlabel('X')
    ax1.set_ylabel('Y')
    ax1.set_zlabel('Z')

    # Subplot 2: Colored by Z value
    ax2 = fig.add_subplot(122, projection='3d')
    p = ax2.scatter(points[:, 0], points[:, 1], points[:, 2], s=1, c=points[:, 2], cmap='jet')
    fig.colorbar(p, ax=ax2)
    ax2.set_title('Points Colored by Z value')
    ax2.set_xlabel('X')
    ax2.set_ylabel('Y')
    ax2.set_zlabel('Z')

    plt.show()
```

```
import random

num_of_samples = 1
for i in range(num_of_samples):
    category = random.choice(categories)
    sample_files = os.listdir(os.path.join(dataset_path, category))
    sample_file_path = os.path.join(dataset_path, category, random.choice(sample_files))
    points = load_txt(sample_file_path)
    print(f"Loaded point cloud of {category}")

    # Visualize the loaded point cloud with faces
    visualize_point_cloud(points)
```

برای سریع تر شدن کار تعداد epoch را ۱۰ در نظر گرفتیم اما تعداد بالاتر نیز برای افزایش دقت میتوان داشت.

Command for train and result of it:

```
!python train_cls_conv.py --model pointconv_modelnet40 --normal --epoch 10
```

```
The size of train data is 9843
The size of test data is 2468
No existing model, starting training from scratch...
Epoch 1 (1/10):
100% 308/308 [03:46<00:00,  1.36it/s]
Train Accuracy: 0.178849
 Loss: 3.133626
 Test Accuracy: 0.200162   ***  Best Accuracy: 0.000000
Epoch 2 (2/10):
100% 308/308 [03:40<00:00,  1.39it/s]
Train Accuracy: 0.303037
 Loss: 2.641269
 Test Accuracy: 0.335900   ***  Best Accuracy: 0.000000
Epoch 3 (3/10):
100% 308/308 [03:40<00:00,  1.39it/s]
Train Accuracy: 0.427498
 Loss: 2.338631
 Test Accuracy: 0.551864   ***  Best Accuracy: 0.000000
Epoch 4 (4/10):
100% 308/308 [03:44<00:00,  1.37it/s]
Train Accuracy: 0.552402
 Loss: 1.801127
 Test Accuracy: 0.641815   ***  Best Accuracy: 0.000000
Epoch 5 (5/10):
100% 308/308 [03:47<00:00,  1.35it/s]
Train Accuracy: 0.630361
 Loss: 1.777007
 Test Accuracy: 0.707050   ***  Best Accuracy: 0.000000
Epoch 6 (6/10):
100% 308/308 [03:47<00:00,  1.35it/s]
Train Accuracy: 0.662428
 Loss: 0.936511
 Test Accuracy: 0.779579   ***  Best Accuracy: 0.000000
```

همانطور که میبینید دقت در هر epoch درحال تغییر است و بطور کل روند صعودی دارد پس از انجام هر مرحله که در نهایت پس از مرحله دهم به دقت زیر میرسیم:

Saving model....

Loss: 0.753525

Test Accuracy: 0.857374 *** Best Accuracy: 0.857374

Best Accuracy: 0.857374

سپس بر روی مدل train شده، evaluation را انجام میدهیم:

```
# After training, evaluate the model (adjust the checkpoint path as needed)
!python eval_cls_conv.py --checkpoint ./checkpoints/checkpoint.pth --normal
```

```
The size of test data is 2468
Load CheckPoint...
Start evaluating...
100% 78/78 [01:11<00:00,  1.10it/s]
Total Accuracy: 0.925080
```

پیاده سازی روش Pointnet برای مقایسه با Pointconv:

برای این کار فایل train_pointnet و همچنین مدل جدید مربوط به pointnet را اضافه کردیم که بتوانیم تست را بر روی آنها انجام دهیم.

نتایج را برای آن مشاهده میکنیم:

```
The size of train data is 9843
The size of test data is 2468
No existing model, starting training from scratch...
Epoch 1 (1/10):
100% 308/308 [01:25<00:00,  3.59it/s]
Train Accuracy: 0.638008
 Loss: 1.083586
 Test Accuracy: 0.656402   ***  Best Accuracy: 0.000000
Epoch 2 (2/10):
100% 308/308 [01:26<00:00,  3.55it/s]
Train Accuracy: 0.778505
 Loss: 0.675164
 Test Accuracy: 0.723258   ***  Best Accuracy: 0.000000
Epoch 3 (3/10):
100% 308/308 [01:32<00:00,  3.32it/s]
Train Accuracy: 0.818443
 Loss: 0.513181
 Test Accuracy: 0.796191   ***  Best Accuracy: 0.000000
Epoch 4 (4/10):
100% 308/308 [01:35<00:00,  3.22it/s]
Train Accuracy: 0.827644
 Loss: 0.419523
 Test Accuracy: 0.820097   ***  Best Accuracy: 0.000000
Epoch 5 (5/10):
100% 308/308 [01:34<00:00,  3.24it/s]
Train Accuracy: 0.842933
 Loss: 0.399680
 Test Accuracy: 0.853323   ***  Best Accuracy: 0.000000
Epoch 6 (6/10):
100% 308/308 [01:35<00:00,  3.21it/s]
Train Accuracy: 0.853688
 Loss: 0.500857
 Test Accuracy: 0.866694   ***  Best Accuracy: 0.000000
Epoch 7 (7/10):
100% 308/308 [01:34<00:00,  3.24it/s]
Train Accuracy: 0.866643
Saving model....
 Loss: 0.320577
 Test Accuracy: 0.861426   ***  Best Accuracy: 0.861426
Epoch 8 (8/10):
100% 308/308 [01:29<00:00,  3.46it/s]
Train Accuracy: 0.871177
 Loss: 0.174182
 Test Accuracy: 0.810373   ***  Best Accuracy: 0.861426
Epoch 9 (9/10):
100% 308/308 [01:29<00:00,  3.44it/s]
Train Accuracy: 0.873441
Saving model....
 Loss: 0.544078
 Test Accuracy: 0.861831   ***  Best Accuracy: 0.861831
Epoch 10 (10/10):
100% 308/308 [01:27<00:00,  3.52it/s]
Train Accuracy: 0.883421
```

و همچنین بعد از evaluation داریم:

```
# After training, evaluate the model (adjust the checkpoint path as needed)
!python eval_cls_conv.py --checkpoint ./checkpoints/checkpoint.pth --normal
```

```
The size of test data is 2468
Load CheckPoint...
Start evaluating...
100% 78/78 [00:57<00:00,  1.36it/s]
Total Accuracy: 0.925080
```

حال بطور کلی به مقایسه این روش با PointConv میپردازیم:

مقایسه PointConv و PointNet از جنبه‌های مختلف مزایا و معایب خود را دارند. در حالی که به طور کلی PointConv به دلیل توانایی آن در درک ساختارهای هندسی محلی ابرنقاط، قدرتمندتر در نظر گرفته می‌شود، عملکرد واقعی می‌تواند بسته به مجموعه داده‌ها، جزئیات پیاده‌سازی و شرایط خاص آموزش متفاوت باشد.

PointNet

سادگی:

- PointNet از لحاظ معماری ساده‌تر است. این مدل ابرنقاط خام را با استفاده از MLPs (شبکه‌های عصبی چند لایه) مستقیماً پردازش می‌کند.
- فهم و پیاده‌سازی آن آسان‌تر است.

سرعت:

- PointNet به دلیل معماری ساده‌تر و محاسبات کمتر معمولاً سریع‌تر است.
- مناسب برای شرایطی که نتایج سریع بدون نیاز به محاسبات سنگین مورد نیاز است.

کارایی حافظه:

- PointNet از لحاظ حافظه کارآمدتر است زیرا نیازی به محاسبه ویژگی‌های محلی یا عملیات کانولوشنی ندارد.
- مناسب برای برنامه‌هایی با منابع محاسباتی محدود.

عملکرد:

- PointNet ممکن است در درک ویژگی‌های هندسی محلی ضعیف باشد زیرا هر نقطه را به صورت مستقل پردازش کرده و سپس اطلاعات کلی را تجمیع می‌کند.

- عملکرد آن ممکن است در وظایفی که نیاز به درک ساختارهای محلی دقیق دارند، پایین‌تر باشد.

PointConv

سادگی:

- PointConv پیچیده‌تر است زیرا PointNet را با عملیات کانولوشنی بر روی ابرنقاط برای درک ویژگی‌های محلی گسترش می‌دهد.
- پیاده‌سازی آن چالش‌برانگیزتر است به دلیل نیاز به تعریف و محاسبه نواحی محلی و فیلترهای کانولوشنی.

سرعت:

- PointConv به دلیل عملیات کانولوشنی اضافی و نیاز به محاسبه همسایگی‌های محلی برای هر نقطه، کندتر است.
- نیاز به منابع محاسباتی و زمان بیشتری دارد.

کارایی حافظه:

- PointConv در مقایسه با PointNet از لحاظ حافظه کارآمدی کمتری دارد زیرا ویژگی‌های محلی و فیلترهای کانولوشنی را ذخیره می‌کند.
- نیاز به حافظه بیشتری دارد که می‌تواند محدودیتی برای ابرنقاط بسیار بزرگ یا محیط‌های با منابع محدود باشد.

عملکرد:

- PointConv معمولاً در وظایفی که نیاز به درک ساختارهای هندسی محلی دارند، بهتر عمل می‌کند به دلیل توانایی آن در درک ویژگی‌های محلی دقیق.
- برای وظایف مثل طبقه‌بندی و بخش‌بندی ابرنقاط که نیاز به زمینه محلی دارند، مقاوم‌تر است.

مقایسه با pointnet از جنبه‌های مختلف

1. سادگی:

- PointNet ساده‌تر و پیاده‌سازی آن آسان‌تر است.

- PointConv پیچیده‌تر است اما درک بهتری از ساختارهای محلی فراهم می‌کند.

2. کارایی حافظه:

- PointNet از لحاظ حافظه کارآمدتر است و مناسب برای محیط‌هایی با منابع محدود است.
- PointConv به حافظه بیشتری نیاز دارد به دلیل محاسبات ویژگی‌های محلی.

3. سرعت:

- PointNet سریع‌تر است و مناسب برای برنامه‌هایی که نتایج سریع نیاز دارند.
- PointConv کندتر است اما ویژگی‌های محلی بهتری استخراج می‌کند.

4. عملکرد:

- PointConv به طور کلی در درک ویژگی‌های هندسی محلی بهتر عمل می‌کند و برای وظایف پیچیده مناسب‌تر است.
- PointNet ممکن است در وظایفی که نیاز به زمینه محلی دقیق دارند عملکرد کمتری داشته باشد اما برای وظایف ساده‌تر کافی است.

نتیجه‌گیری

در حالی که PointNet سادگی، سرعت و کارایی حافظه را ارائه می‌دهد، PointConv معمولاً عملکرد بهتری برای وظایفی که نیاز به درک دقیق ساختارهای هندسی محلی دارند، دارد. انتخاب بین PointNet و PointConv باید بر اساس نیازهای خاص برنامه، شامل نیاز به دقت، منابع محاسباتی موجود و پیچیدگی وظیفه باشد.

در مورد فعلی تست شده، اگرچه PointNet دقت بالاتری در مجموعه داده‌های فعلی به دست آورده است، PointConv ممکن است برای برنامه‌هایی که نیاز به استخراج ویژگی‌های محلی قوی دارند، ترجیح داده شود. با این حال، انتخاب نهایی به مبادلاتی که مایلید بین پیچیدگی، سرعت، استفاده از حافظه و عملکرد انجام دهید بستگی دارد. هرچند این که تعداد epoch های کمتری به دلیل محدودیت زمانی تست شد هم بی تاثیر نیست.

برای مقایسه با روش های دیگر بطور کلی میتوان به مقاله زیر مراجعه کرد:
https://www.researchgate.net/figure/Performance-comparison-of-different-network-models-on-ModelNet40-dataset_tbl2_369399640

مقایسه رویکرد PointConv با DGCNN، PointNet++، PointNet و PVCNN در زمینه حافظه، پیچیدگی، سرعت و عملکرد

حافظه

- **PointNet**: از آنجایی که PointNet یک معماری نسبتاً ساده است و تنها از چند لایه همگرا استفاده می‌کند، مصرف حافظه کمتری نسبت به مدل‌های پیچیده‌تر دارد.
- **PointNet++**: PointNet++ با استفاده از عملیات نمونه‌گیری سلسله‌مراتبی و گروه‌بندی نقطه‌ها، مصرف حافظه بیشتری نسبت به PointNet دارد، اما همچنان بهینه است.
- **DGCNN**: DGCNN با استفاده از گراف و توجه به همسایگی محلی، مصرف حافظه بیشتری دارد. به‌خصوص در پردازش گراف‌های بزرگ، مصرف حافظه افزایش می‌یابد.
- **PVCNN**: PVCNN با استفاده از کانولوشن‌های وکسیل، مصرف حافظه بیشتری دارد، اما به دلیل بهره‌گیری از فضای سه‌بعدی، مصرف حافظه بهینه‌تری نسبت به DGCNN دارد.
- **PointConv**: PointConv با ترکیب کانولوشن نقطه‌ای و شبکه‌های عصبی وزنی، مصرف حافظه متوسطی دارد و بهینه‌تر از DGCNN و PVCNN عمل می‌کند.

پیچیدگی

- **PointNet**: پیچیدگی محاسباتی PointNet پایین است زیرا فقط از چند لایه همگرا و عملیات حداکثر استفاده می‌کند.
- **PointNet++**: با افزودن پیچیدگی‌های نمونه‌گیری و گروه‌بندی، پیچیدگی محاسباتی PointNet++ افزایش یافته است.
- **DGCNN**: به دلیل ساخت و پردازش گراف، پیچیدگی محاسباتی بالایی دارد.
- **PVCNN**: پیچیدگی محاسباتی متوسطی دارد و بهینه‌سازی‌هایی را برای کاهش پیچیدگی در فضای سه‌بعدی انجام داده است.
- **PointConv**: پیچیدگی محاسباتی PointConv متوسط است و با استفاده از شبکه‌های عصبی وزنی، بهینه‌سازی‌هایی را در پردازش نقطه‌ای انجام داده است.

سرعت

- **PointNet**: سرعت پردازش بالایی دارد و می‌تواند به‌سرعت نقطه‌ها را پردازش کند.
- **PointNet++**: سرعت پردازش کمتری نسبت به PointNet دارد به دلیل عملیات‌های پیچیده‌تر نمونه‌گیری و گروه‌بندی.
- **DGCNN**: سرعت پردازش پایین‌تری دارد به دلیل ساخت و پردازش گراف.
- **PVCNN**: سرعت پردازش متوسطی دارد و بهینه‌سازی‌هایی را برای افزایش سرعت در فضای سه‌بعدی انجام داده است.
- **PointConv**: سرعت پردازش PointConv بالاتر از DGCNN و PVCNN است و با استفاده از بهینه‌سازی‌های شبکه‌های عصبی وزنی، سرعت قابل‌توجهی دارد.

عملکرد

- **PointNet**: عملکرد مناسبی در تشخیص اشیاء و دسته‌بندی نقطه‌ها دارد، اما در برخی موارد دقیق نیست.
- **PointNet++**: عملکرد بهتری نسبت به PointNet دارد به دلیل توجه به همسایگی‌های محلی و سلسله‌مراتبی.
- **DGCNN**: عملکرد بسیار خوبی دارد به دلیل بهره‌گیری از گراف و توجه به ارتباطات محلی بین نقطه‌ها.
- **PVCNN**: عملکرد مناسبی دارد و بهینه‌سازی‌هایی را برای کاهش پیچیدگی و افزایش دقت انجام داده است.
- **PointConv**: عملکرد بسیار خوبی دارد و با استفاده از کانولوشن نقطه‌ای و شبکه‌های عصبی وزنی، دقت بالایی را در تشخیص و دسته‌بندی نقطه‌ها ارائه می‌دهد.

نتیجه‌گیری

در مجموع، PointConv با ترکیب ویژگی‌های کانولوشن نقطه‌ای و شبکه‌های عصبی وزنی، تعادلی بین مصرف حافظه، پیچیدگی محاسباتی، سرعت و عملکرد ارائه می‌دهد. این مدل در مقایسه با مدل‌های دیگر مانند DGCNN ،PointNet++ ،PointNet و PVCNN عملکرد بهتری دارد و می‌تواند به‌عنوان یک رویکرد بهینه برای پردازش ابر نقطه‌ها در نظر گرفته شود.

ایده جدید

در این پروژه، چند روش داده‌افزایی (Data Augmentation) پیشرفته برای بهبود مدل PointConv اضافه شده‌اند. این روش‌ها به‌طور تصادفی و با احتمال کم ثابت (0.1) اعمال می‌شوند تا زمان آموزش در هر دوره (epoch) به‌طور قابل توجهی افزایش نیابد.

روش‌های داده‌افزایی افزوده شده

1. **random_rotation_point_cloud**: این روش نقطه‌ابری را به‌طور تصادفی حول محور z می‌چرخاند.
2. **random_translation_point_cloud**: این روش نقطه‌ابری را به‌طور تصادفی در فضا ترجمه می‌کند.
3. **random_noise_point_cloud**: این روش نویز گوسی به هر نقطه در نقطه‌ابری اضافه می‌کند.

پیاده‌سازی

کدهای مربوطه در فایل provider.py اضافه شده اند.

```python
def random_rotation_point_cloud(batch_data):
    """ Randomly rotate the point clouds to augment the dataset
        rotation is per shape based along up direction
        Input:
            BxNx3 or BxNx6 array, original batch of point clouds
        Return:
            BxNx3 or BxNx6 array, rotated batch of point clouds
    """
    rotated_data = np.zeros(batch_data.shape, dtype=np.float32)
    for k in range(batch_data.shape[0]):
        rotation_angle = np.random.uniform() * 2 * np.pi
        cosval = np.cos(rotation_angle)
        sinval = np.sin(rotation_angle)
        rotation_matrix = np.array([[cosval, -sinval, 0],
                                    [sinval, cosval, 0],
                                    [0, 0, 1]])
        shape_pc = batch_data[k, :, :3]
        rotated_data[k, :, :3] = np.dot(shape_pc, rotation_matrix)
        if batch_data.shape[2] > 3:
            rotated_data[k, :, 3:] = batch_data[k, :, 3:]
    return rotated_data

def random_translation_point_cloud(batch_data, shift_range=0.1):
    """ Randomly translate the point clouds to augment the dataset
        translation is per shape
        Input:
            BxNx3 or BxNx6 array, original batch of point clouds
        Return:
            BxNx3 or BxNx6 array, translated batch of point clouds
    """
    translated_data = np.zeros(batch_data.shape, dtype=np.float32)
    for k in range(batch_data.shape[0]):
        translation = np.random.uniform(-shift_range, shift_range, 3)
        translated_data[k, :, :3] = batch_data[k, :, :3] + translation
        if batch_data.shape[2] > 3:
            translated_data[k, :, 3:] = batch_data[k, :, 3:]
    return translated_data

def random_noise_point_cloud(batch_data, sigma=0.01, clip=0.05):
    """ Randomly add Gaussian noise to each point in the point cloud
        Input:
            BxNx3 or BxNx6 array, original batch of point clouds
        Return:
            BxNx3 or BxNx6 array, point clouds with added Gaussian noise
    """
    noise = np.clip(sigma * np.random.randn(*batch_data.shape), -clip, clip)
    batch_data += noise
    return batch_data
```

و در فایل train_cls_conv در بخش training، استفاده شده اند.

```
for batch_id, data in tqdm(enumerate(trainDataLoader, 0), total=len(trainDataLoader), smoothing=0.9):
            points, target = data
            points = points.data.numpy()

            if random.random() < AUGMENTATION_PROB:
                points = random_rotation_point_cloud(points)
            if random.random() < AUGMENTATION_PROB:
                points = random_translation_point_cloud(points)
            if random.random() < AUGMENTATION_PROB:
                points = random_noise_point_cloud(points)
```

با اعمال این روش‌های داده‌افزایی پیشرفته به‌طور تصادفی و با احتمال کم، زمان آموزش به‌طور قابل توجهی کاهش یافته و مدل توانسته از مزایای داده‌افزایی بهره‌مند شود بدون اینکه بار محاسباتی زیادی به فرآیند آموزش اضافه شود.

اهداف اصلی از Data augmentation:

1. جلوگیری از بیش‌برازش (Overfitting)

داده‌افزایی با بزرگ کردن مصنوعی مجموعه داده‌های آموزشی، به جلوگیری از بیش‌برازش کمک می‌کند. با اعمال تبدیلات تصادفی به داده‌های آموزشی، مدل با سناریوهای گسترده‌تری روبرو می‌شود. این احتمال را کاهش می‌دهد که مدل داده‌های آموزشی را حفظ کند و نتواند به داده‌های جدید و نادیده تعمیم یابد.

2. بهبود تعمیم‌دهی (Generalization)

با داده‌افزایی مجموعه داده‌های آموزشی، مدل می‌آموزد که به تغییرات خاصی در داده‌های ورودی بی‌تفاوت باشد. به عنوان مثال، از طریق چرخش‌های تصادفی، ترجمه‌ها و مقیاس‌گذاری، مدل یاد می‌گیرد که اشیاء را بدون توجه به جهت‌گیری، موقعیت یا اندازه آنها تشخیص دهد. این قابلیت تعمیم‌دهی مدل را از داده‌های آموزشی به موقعیت‌های واقعی افزایش می‌دهد.

3. افزایش استحکام (Robustness)

داده‌افزایی تغییراتی را معرفی می‌کند که مدل ممکن است در سناریوهای واقعی با آنها مواجه شود، و این باعث می‌شود مدل به این تغییرات مقاوم‌تر شود. به عنوان مثال، افزودن نویز به داده‌ها می‌تواند مدل را نسبت به ورودی‌های نویزی تحمل‌پذیرتر کند.

4. شبیه‌سازی تغییرات دنیای واقعی

در بسیاری از کاربردها، شرایط جمع‌آوری داده‌ها می‌تواند به طور قابل توجهی تغییر کند. داده‌افزایی به شبیه‌سازی این تغییرات کمک می‌کند و اطمینان می‌دهد که مدل در شرایط مختلف عملکرد خوبی دارد. این امر به ویژه در زمینه‌هایی مانند بینایی کامپیوتر مهم است، که در آن نور، جهت‌گیری و پس‌زمینه می‌توانند تغییر کنند.

```
Epoch 1 (1/10):
100% 308/308 [03:45<00:00,  1.37it/s]
Train Accuracy: 0.204449
 Loss: 3.646400
 Test Accuracy: 0.295381   ***  Best Accuracy: 0.000000
Epoch 2 (2/10):
100% 308/308 [03:43<00:00,  1.38it/s]
Train Accuracy: 0.350863
 Loss: 2.857113
 Test Accuracy: 0.425446   ***  Best Accuracy: 0.000000
Epoch 3 (3/10):
100% 308/308 [03:42<00:00,  1.38it/s]
Train Accuracy: 0.453542
 Loss: 2.537223
 Test Accuracy: 0.587115   ***  Best Accuracy: 0.000000
Epoch 4 (4/10):
100% 308/308 [03:39<00:00,  1.40it/s]
Train Accuracy: 0.549358
 Loss: 1.723738
 Test Accuracy: 0.677066   ***  Best Accuracy: 0.000000
Epoch 5 (5/10):
100% 308/308 [03:40<00:00,  1.40it/s]
Train Accuracy: 0.607228
 Loss: 2.164554
 Test Accuracy: 0.747974   ***  Best Accuracy: 0.000000
Epoch 6 (6/10):
100% 308/308 [03:43<00:00,  1.38it/s]
Train Accuracy: 0.649473
 Loss: 1.131599
 Test Accuracy: 0.753241   ***  Best Accuracy: 0.000000
```

```
Epoch 7 (7/10):
100% 308/308 [03:41<00:00,  1.39it/s]
Train Accuracy: 0.674267
Saving model....
 Loss: 1.175732
 Test Accuracy: 0.795381   ***  Best Accuracy: 0.795381
Epoch 8 (8/10):
100% 308/308 [03:41<00:00,  1.39it/s]
Train Accuracy: 0.702949
Saving model....
 Loss: 0.709750
 Test Accuracy: 0.826580   ***  Best Accuracy: 0.826580
Epoch 9 (9/10):
100% 308/308 [03:40<00:00,  1.40it/s]
Train Accuracy: 0.726221
 Loss: 0.548715
 Test Accuracy: 0.819692   ***  Best Accuracy: 0.826580
Epoch 10 (10/10):
100% 308/308 [03:40<00:00,  1.40it/s]
Train Accuracy: 0.737173
Saving model....
 Loss: 0.524892
 Test Accuracy: 0.833468   ***  Best Accuracy: 0.833468
Best Accuracy: 0.833468
```