研究生《深度学习》课程

实验报告

实验名称：	综合实验
姓　　名：	唐麒
学　　号：	21120299
上课类型：	专业课
日　　期：	2022 年 12 月 19 日

一、实验内容

本实验报告包括的实验有出租车流量预测和顶会论文复现，其中出租车流量预测属于时空数据学习领域的任务，而顶会论文复现选取的是计算机视觉领域的任务——视频实例分割。通过本次实验，学生将对卷积神经网络的原理及实现、循环神经网络及其变体的设计和实现有更加深刻的理解。

1.1 出租车流量预测

车流量预测任务是一个回归任务，旨在根据区域历史的车流量情况来预测其未来某一段时间的车流量情况 。使用的数据为纽约市出租车流量数据。输入为纽约市各区域的历史车流量时间序列，输出为对应各区域的未来车流量的预测值。要求同学们以提高在测试集上的效果为目标，自己根据数据特点及需要进行数据预处理以及模型设计，本任务不对模型的设计和选择进行限制，但根据数据特点可结合 CNN 与 RNN 的特点进行模型设计。

本任务要求首先将数据进行归一化处理并对数据进行滑动窗口划分，具体划分要求为使用历史 6 个时间步预测未来一个时间步，模型的输出应该为纽约 200 个区域的各区域未来半小时的出入流量。在计算评价指标 MAE 和 RMSE 时应将数据反归一化后再计算。

1.2 顶会论文复现

本实验选取的论文是 SeqFormer: Sequential Transformer for Video Instance Segmentation (ECCV 2022 Oral)，虽然 ECCV 是 CCF 推荐的 B 类会议，但其作为计算机视觉领域的三大会议，收录的工作也是备受关注。本文聚焦的任务是视频实例分割，该任务在一篇 2019 年 ICCV 的工作中首次提出，集视频目标检测、分割和追踪任务于一体，因而更加具有挑战性，一经提出便获得了广泛关注。除本篇论文之外，该作者还同时在 ECCV 2022 中发表了另一篇利用相同网络架构的研究工作，该研究在研究在线视频实例分割方法和离线视频实例分割方法的性能差异后，在相同网络架构的基础上增加了特殊设计的对比学习模式，从而实现了在线方法性能对离线方法性能的超越。综上本实验选择该论文开源的代码进行复现和实验探究。

二、实验设计

本次实验所包含的内容包括自行设计的实验内容和对开源代码的复现，本部分主要对出租车流量预测的模型设计进行介绍，针对开源代码的复现将在实验过程部分详细介绍。

对于时序数据的训练和测试，很容易想到的便是利用 RNN 及其变体 LSTM 和 GRU。但对于本文所使用的数据，某个地点的出租车流量不仅与其邻近时间点有关、还与其邻近节点的出租车流量有关，因此可以考虑利用卷积操作适合处理局部相关性强信号的能力。为此本实验借鉴特征融合的思想，在 LSTM 的基础上添加了 3D 卷积来提取额外的特征并与 LSTM 提取到的特征进行融合，最后进行预测。综上，本文分别在经典的 LSTM、GRU 和结合 3D 卷积的 LSTM、结合 3D 卷积的 GRU 四种网络架构上进行实验，选取效果最好的模型，并进一步对超参数进行调整，从而获得更好的预测精度。网络架构图如图 1 所示。

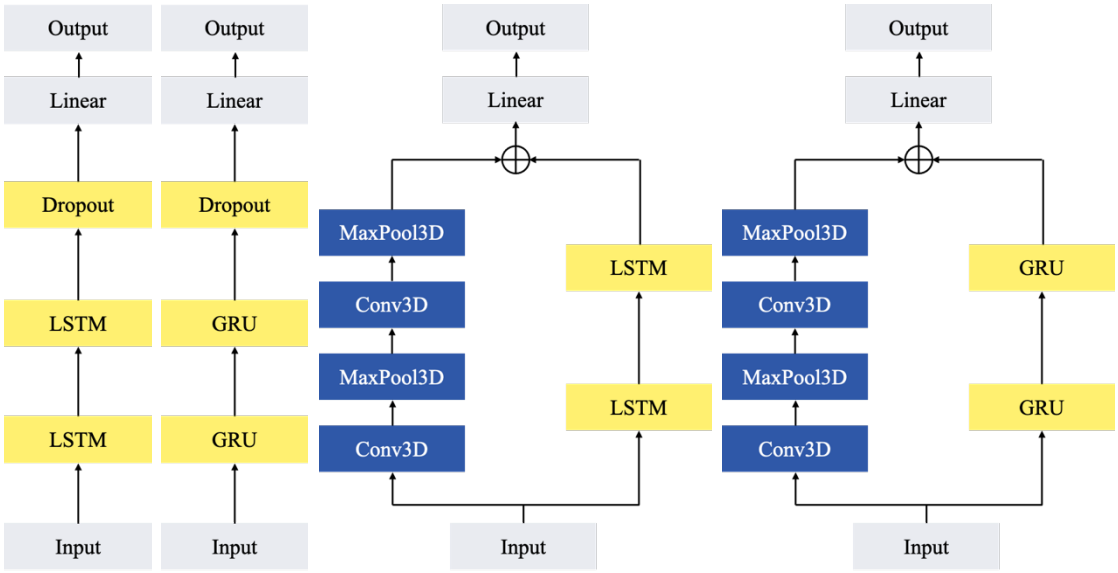

图 1 出租车流量预测网络架构图：(a) LSTM, (b) GRU, (c) 结合 3D 卷积的 LSTM, (d) 结合 3D 卷积的 GRU.

三、实验环境及实验数据集

实验所用的开发环境如表 1 所示。

表 1 实验环境

开发环境	版本
操作系统	macOS Big Sur 11.5.1
开发工具	Jupyter Notebook
Python	3.8.2
PyTorch	1.8.0 (CPU)

本次实验所涉及的数据集包括：纽约出租车流量数据集和 YouTube VIS 2019。纽约出租车流量数据集和 YouTube VIS 2019 分别对应了出租车流量预测和顶会论文复现。

纽约出租车流量数据集，时间跨度为从 2015 年 1 月 1 日到 2015 年 3 月 1 日。数据处理成为网格流量数据时的时间间隔设定为 30 分钟。后 20 天数据被划定为测试集，其余数据为训练集。以训练集为例，其 shape=(1920*10*20*2) 代表有 1920 个时间段，10*20 个区域，2 个特征分别为区域的入流量与出流量。

YouTube VIS 是在 YouTube VOS 大型视频对象分割数据集基础上建立的数据集，该数据集及其之后的更新版本也是作为了视频实例分割研究的 benchmark。YouTube VOS 由 4453 个高分辨率 YouTube 视频和 94 个常见对象类别组成。在每个视频中，通过以 30fps 的帧率每隔 5 帧手动跟踪对象边界来标记几个对象。每个视频的长度约为 3 到 6 秒。YouTube VIS 从 94 个类别标签中选择 40 个常见的类别标签作为类别集，然后采样了包含来自 40 个类别的对象大约 2.9k 的视频，并通过人工仔细标记属于这些视频中类别集的对象。数据集中每个类别的唯一对象的分布如图 2 所示。

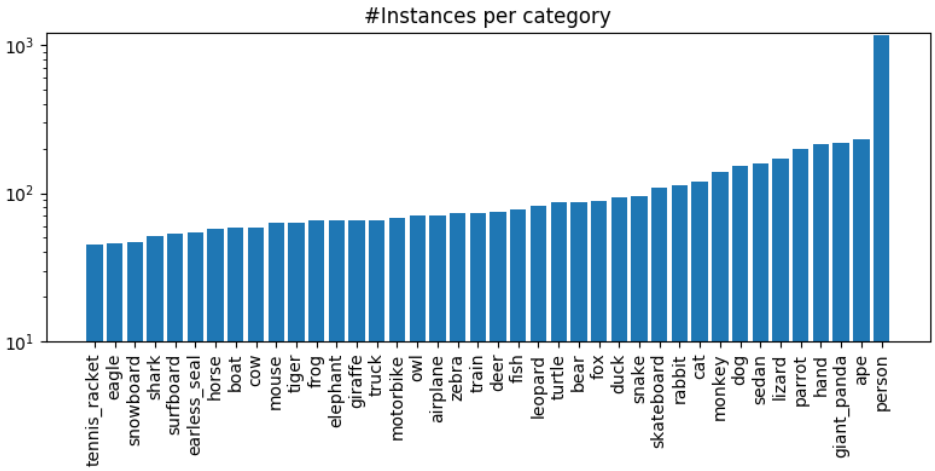

图 2 YouTube VIS 2019 类别分布统计

四、实验过程

本小节将对实验出租车流量预测的具体内容进行详细说明。

首先，以训练集为例对数据处理的方法和代码进行介绍和展示。

- 读取数据 data = np.load(data_path)['volume'] 该语句的作用是读取数据，原始数据的形状是(1920,10,20,2)；
- 对数据进行归一化处理，其中 NYCSTDNDataset 类里包含了这一处理过程。具体来说，记录原始数据中的最大值和最小值为 max_val 和 min_val，然后将 data 减去最小值的差再除以最大值和最小值的差便可归一化。其中 data = (data - self.min_val) / (self.max_val - self.min_val) 是其执行语句；
- 将数据进行划分序列，其中 slidingWindow 函数用来将一个长序列划分为短序列。具体来说，定义一个 slidingWindow 函数以封装滑动窗口划分序列的过程，接口参数有 seqs 和 size，seqs 为要被划分的原始序列，size 为滑动窗口大小，size=7，即用 6 个历史时间步的历史车流量来预测最后一个时间步的车流量。定义一个列表 result 以保存划分完的结果，在 for 循环里执行划分过程，遍历的次数是原始序列的长度-滑动窗口的大小+1，取出原始序列中的一个切片，该切片中第一维度的值随着遍历次数的变化而变化，其长度为 size，其他维度与原始序列相同。每次循环都将切片保存在 result 中，当所有循环结束，划分序列便完成。数据的形状由(1920,10,20,2) 变为 (1914, 7, 10, 20, 2)；
- 改变数据的形状 将数据的形状由 (1914, 7, 10, 20, 2)变为 (1914, 7, 400)。

代码实现如下：

```
class NYCSTDNDataset():
    def __init__(self,data_path,window_size=7) -> None:
        self.window_size = window_size
        self.data = torch.from_numpy(self.loading(data_path)).float()
    def loading(self,data_path):
        data = np.load(data_path)['volume']
        self.max_val,self.min_val = np.max(data),np.min(data)
```

```
            dataset = slidingWindow(data,self.window_size)
            dataset = np.array(dataset).transpose(0,1,4,2,3) # (1914, 7, 2, 10, 20)
            dataset = dataset.reshape(dataset.shape[0],dataset.shape[1],-1)
            dataset = (dataset - self.min_val) / (self.max_val - self.min_val)
            return dataset
    def denormalize(self,x):
            return x * (self.max_val - self.min_val) + self.min_val
def slidingWindow(seqs,size):
    """
    seqs: ndarray sequence, shape(seqlen,area_nums,2)
    size: sliding window size
    """
    result = []
    for i in range(seqs.shape[0] - size + 1):
            result.append(seqs[i:i + size,:,:,:]) #(7, 10, 20, 2)
    return result
```

为了利用图表对 Loss、RMSE、MAE、MAPE 变化进行可视化展示，定义了绘图函数，其中 MSE、MAE 引入第三方包的实现，而 RMSE 由 MSE 开方所得，MAPE 需要自行构建。

```
def drawPlot(heights,fname,ylabel):
    """
    功能：绘制训练集上的准确率和测试集上的 loss 和 acc 变化曲线
    heights: 纵轴值列表
    fname：保存的文件名
    """
    plt.figure(figsize=(9, 6))
    x = [i for i in range(1,len(heights[0]) + 1)]
    # 绘制训练集和测试集上的 loss 变化曲线子图
    plt.xlabel("epoch")
    # 设置横坐标的刻度间隔
    plt.xticks([i for i in range(0,len(heights[0]) + 1,5)])
    axe1 = plt.subplot(2,2,1)
    plt.ylabel(ylabel[0])
    axe1.plot(x,heights[0],label="train")
    axe1.plot(x,heights[1],label="test")
    axe1.legend()
    axe2 = plt.subplot(2,2,2)
```

```
        plt.ylabel(ylabel[1])
        axe2.plot(x,heights[2],label="train")
        axe2.plot(x,heights[3],label="test")
        plt.legend()
        axe3 = plt.subplot(2,2,3)
        plt.ylabel(ylabel[2])
        axe3.plot(x,heights[4],label="train")
        axe3.plot(x,heights[5],label="test")
        plt.legend()
        axe4 = plt.subplot(2,2,4)
        plt.ylabel(ylabel[3])
        axe4.plot(x,heights[6],label="train")
        axe4.plot(x,heights[7],label="test")
        plt.legend()
        plt.savefig("images/{}".format(fname))
        plt.show()
def mape(y_true, y_pred):
        y_true, y_pred = np.array(y_true), np.array(y_pred)
        non_zero_index = (y_true > 0)
        y_true = y_true[non_zero_index]
        y_pred = y_pred[non_zero_index]
        mape = np.abs((y_true - y_pred) / y_true)
        mape[np.isinf(mape)] = 0
        return np.mean(mape) * 100
def nextBatch(data,batch_size):
        """
        Divide data into mini-batch
        """
        data_length = len(data)
        num_batches = math.ceil(data_length / batch_size)
        for idx in range(num_batches):
                start_idx = batch_size * idx
                end_idx = min(start_idx + batch_size, data_length)
                yield data[start_idx:end_idx]
```

标准 LSTM、GRU 和结合 3D 卷积的 LSTM、GRU 代码实现如下。

```
class MyLSTM(nn.Module):
        def __init__(self,input_size,hidden_size,output_size,drop_prob):
```

```
            super(MyLSTM,self).__init__()
            self.lstm = nn.LSTM(input_size=input_size,
                                hidden_size=hidden_size,
                                batch_first=True,
                                num_layers=2)
            self.dropout = nn.Dropout(drop_prob)
            self.fc = nn.Linear(hidden_size,output_size)
        def forward(self,x):
            """
            x: (batch,seq,feature)
            """
            x,_ = self.lstm(x)
            x = self.dropout(x)
            x = self.fc(torch.mean(x,dim=1))
            x = F.sigmoid(x)
            return x

class MyGRU(nn.Module):
        def __init__(self,input_size,hidden_size,output_size,drop_prob):
            super(MyGRU,self).__init__()
            self.gru = nn.GRU(input_size=input_size,
                                hidden_size=hidden_size,
                                batch_first=True,
                                num_layers=2)
            self.dropout = nn.Dropout(drop_prob)
            self.fc = nn.Linear(hidden_size,output_size)
        def forward(self,x):
            """
            x: (batch,seq,feature)
            """
            x,_ = self.gru(x)
            x = self.dropout(x)
            x = self.fc(torch.mean(x,dim=1))
            x = F.sigmoid(x)
            return x

class CNNLSTM(nn.Module):
    def __init__(self,in_channel,out_channels,input_size,hidden_size,output_size,drop_prob):
```

```
41              super(CNNLSTM,self).__init__()
42              self.convs = nn.Sequential(
43                  nn.Conv3d(in_channels=in_channel, out_channels=out_channels[0],
44                            kernel_size=(3, 3, 3)),
45                  nn.MaxPool3d(kernel_size=(1, 2, 2), stride=(1, 2, 2)),
46                  nn.Conv3d(in_channels=out_channels[0], out_channels=out_channels[1],
47                            kernel_size=(3, 3, 3)),
48                  nn.MaxPool3d(kernel_size=(2, 2, 2), stride=(2, 2, 2))
49              )
50              self.lstm = nn.LSTM(input_size=input_size,
51                                  hidden_size=hidden_size,
52                                  batch_first=True,
53                                  num_layers=2)
54              self.dropout = nn.Dropout(drop_prob)
55              self.fc = nn.Linear(384 + hidden_size,output_size)
56          def forward(self,x):
57              """
58              x: (batch,depth/seqlen,channels*h*w)
59              """
60              x = x.view(x.shape[0],x.shape[1],2,10,20)
61              cnn_feats = self.convs(x.transpose(1,2))
62              gru_out,_ = self.lstm(x.view(x.shape[0],x.shape[1],-1))
63              gru_feats =torch.mean(gru_out,dim=1)
64              fusion_feats = torch.cat((cnn_feats.view(cnn_feats.shape[0],-1),gru_feats),dim=1)
65              x = self.fc(self.dropout(fusion_feats))
66              return F.sigmoid(x)
67
68  class CNNGRU(nn.Module):
69      def
70  __init__(self,in_channel,out_channels,input_size,hidden_size,output_size,drop_prob):
71              super(CNNGRU,self).__init__()
72              self.convs = nn.Sequential(
73                  nn.Conv3d(in_channels=in_channel, out_channels=out_channels[0],
74                            kernel_size=(3, 3, 3), padding=(1, 1, 1)),
75                  nn.MaxPool3d(kernel_size=(1, 2, 2), stride=(1, 2, 2)),
76                  nn.Conv3d(in_channels=out_channels[0], out_channels=out_channels[1],
77
78                            kernel_size=(3, 3, 3), padding=(1, 1, 1)),
                  nn.MaxPool3d(kernel_size=(2, 2, 2), stride=(2, 2, 2)),
```

```
                nn.Conv3d(in_channels=out_channels[0], out_channels=out_channels[1],
                          kernel_size=(3, 3, 3), padding=(1, 1, 1)),
                nn.Conv3d(in_channels=out_channels[0], out_channels=out_channels[1],
                          kernel_size=(3, 3, 3), padding=(1, 1, 1)),
                nn.MaxPool3d(kernel_size=(2, 2, 2), stride=(2, 2, 2))
            )
            self.gru = nn.GRU(input_size=input_size,
                              hidden_size=hidden_size,
                              batch_first=True,
                              num_layers=2)
            self.dropout = nn.Dropout(drop_prob)
            self.fc = nn.Linear(4 + hidden_size,output_size)
        def forward(self,x):
            """
            x: (batch,depth/seqlen,channels*h*w)
            """
            x = x.view(x.shape[0],x.shape[1],2,10,20)
            cnn_feats = self.convs(x.transpose(1,2))
            gru_out,_ = self.gru(x.view(x.shape[0],x.shape[1],-1))
            gru_feats =torch.mean(gru_out,dim=1)
            fusion_feats = torch.cat((cnn_feats.view(cnn_feats.shape[0],-1),gru_feats),dim=1)
            x = self.fc(self.dropout(fusion_feats))
            return F.sigmoid(x)
```

训练和测试的代码实现如下。

```
def train_one_epoch(model, train_dataset, loss_fn, optimizer):
    model.train()
    iteration = 0
    for batch in nextBatch(shuffle(train_dataset.data), batch_size):
        x, y = batch[:, :-1, :], batch[:, -1, :]
        x, y = x.to(device), y.to(device)
        y_hat = model(x)
        l = loss_fn(y_hat, y)
        optimizer.zero_grad(set_to_none=True)
        l.backward()
        optimizer.step()
        iteration += 1
```

```
            if iteration % 10 == 0:
                print("Iteraion {}, Train Loss: {:.8f}".format(iteration, l.item()))
        train_rmse, train_mae, train_mape, train_loss = test(model, train_dataset, loss_fn)
        return (train_rmse, train_mae, train_mape, train_loss)

def test(model,test_dataset,loss_fn):
        model.eval()
        y_hats = []
        test_l_sum,c = 0,0
        with torch.no_grad():
            for batch in nextBatch(test_dataset.data, batch_size=batch_size):
                x,y = batch[:,:-1,:], batch[:,-1,:]
                x,y = x.to(device),y.to(device)
                y_hat = model(x)
                test_l_sum += loss_fn(y_hat,y).item()
                c += 1
                y_hats.append(y_hat.detach().cpu().numpy())
            y_hats = np.concatenate(y_hats)
        y_true = test_dataset.data[:,-1,:]
        y_hats = test_dataset.denormalize(y_hats)
        y_true = test_dataset.denormalize(y_true)
        y_true = y_true.reshape(y_true.size(0),-1)
        rmse_score,mae_score,mape_score = math.sqrt(mse(y_true, y_hats)), mae(y_true,
        y_hats), mape(y_true, y_hats)
        return (rmse_score,mae_score,mape_score,test_l_sum / c)
```

五、实验结果

在纽约出租车流量数据集保持相同的训练参数（lr = 0.001, batch size = 64, hidden size = 64, epoch = 1000, drop probability = 0.5）对 LSTM、GRU、结合 3D 卷积的 LSTM 和 GRU 训练结果如图 3-6 所示。从图中可以看到，LSTM 和 GRU 作为 RNN 的变体，可以较好地处理时序数据的学习和预测，而与对结合卷积模型的实验效果对比，可以参考表 2。

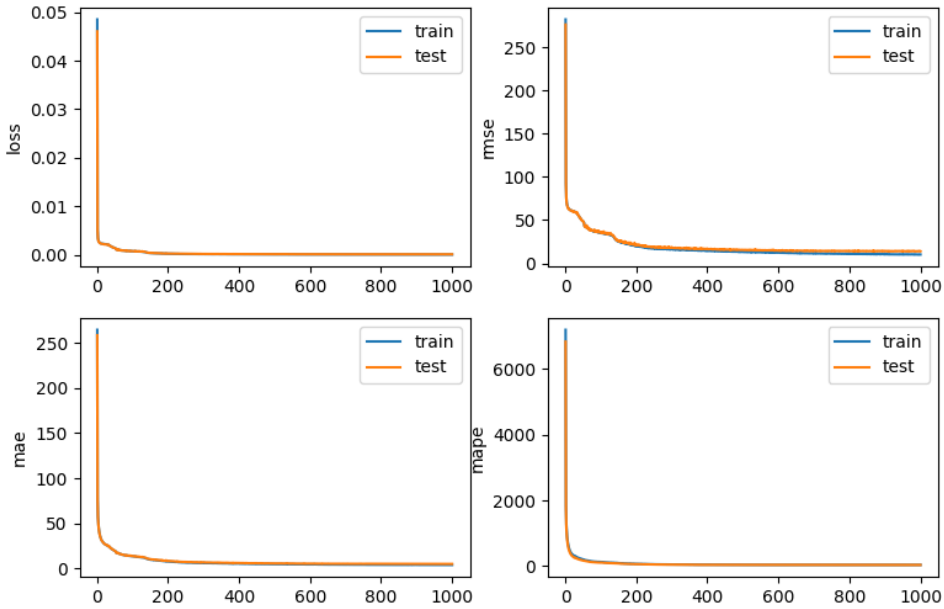

图 3 LSTM 损失函数、RMSE、MAE、MAPE 变化曲线图

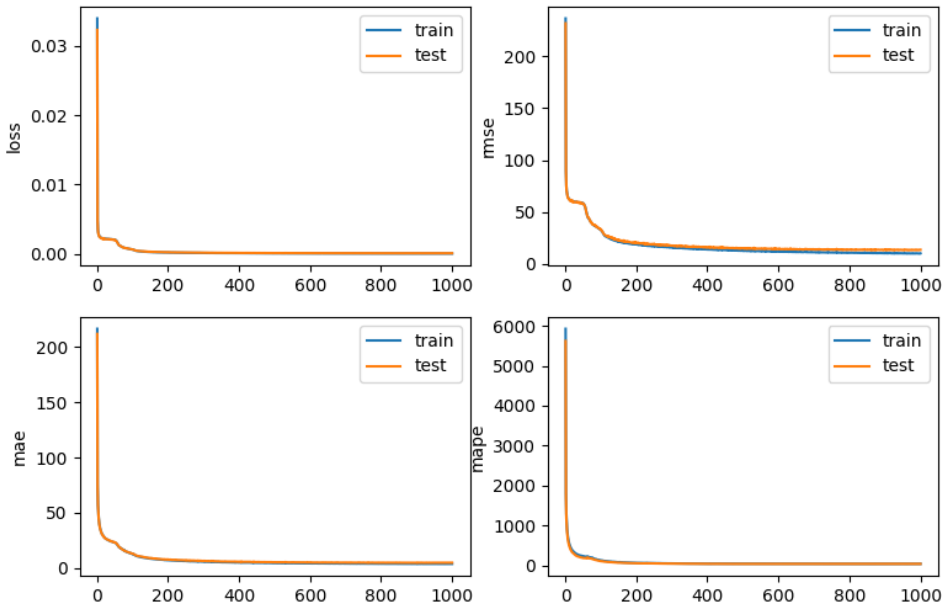

图 4 GRU 损失函数、RMSE、MAE、MAPE 变化曲线图

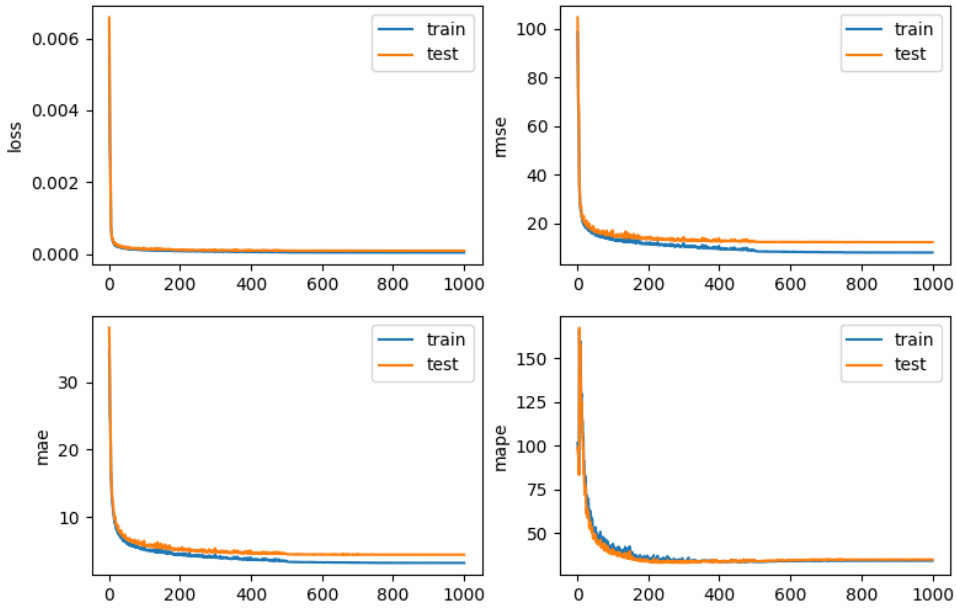

图 5 结合 3D 卷积的 LSTM 损失函数、RMSE、MAE、MAPE 变化曲线图

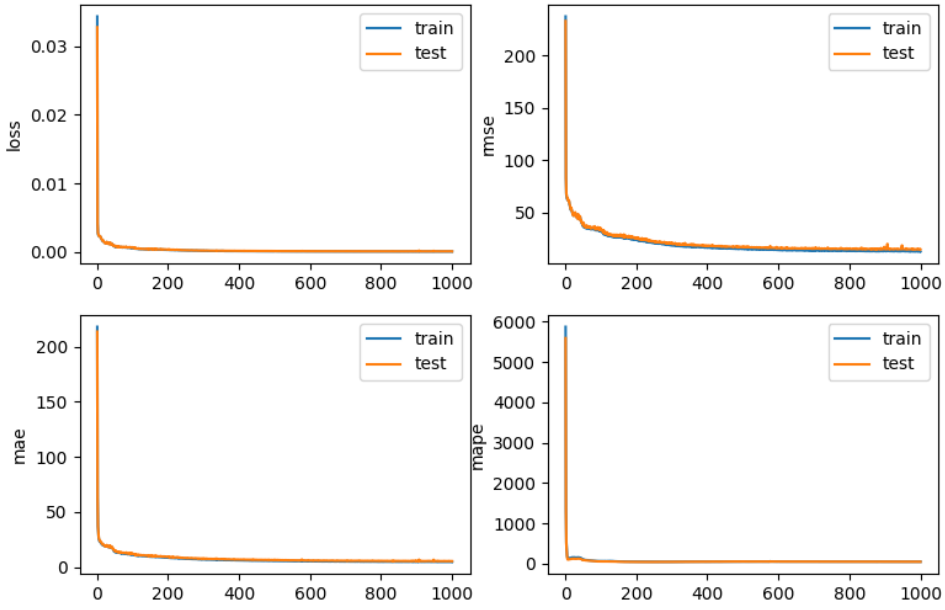

图 6 结合 3D 卷积的 GRU 损失函数、RMSE、MAE、MAPE 变化曲线图

表 2 模型实验结果

模型	Best Epoch	RMSE	MAE	MAPE
LSTM	967	13.73	4.87	35.94
GRU	964	13.34	4.80	37.33
CNN LSTM	548	12.38	4.54	34.00
CNN GRU	983	14.36	5.27	49.98

从表 2 可以看出，相较于其他模型，结合 3D 卷积的 LSTM 可以取得更好的性能表现，而且训练所需的迭代次数更少，因此可以证明除了 LSTM 可以很好地利用数据集中相邻时间节点的有效信息外，卷积也可以提供额外有用的相邻节点的信息。后续的超参数调整实验将在结合 3D 卷积的 LSTM 上进行。除此以外，较少的训练迭代次数即可使该模型取得较好的性能表现，也说明了后续的迭代中，由于不合适的学习率调整策略使得模型训练可能发生了一定的震荡。

下面将分别探究 hidden size、drop probability、学习率调整策略和 LSTM 层数对模型性能的影响。

图 7 和图 8 分别为 hidden size 调整为 128 和 256 的结合 3D 卷积的 LSTM 模型性能变化曲线图，具体的数据对比如表 3 所示。

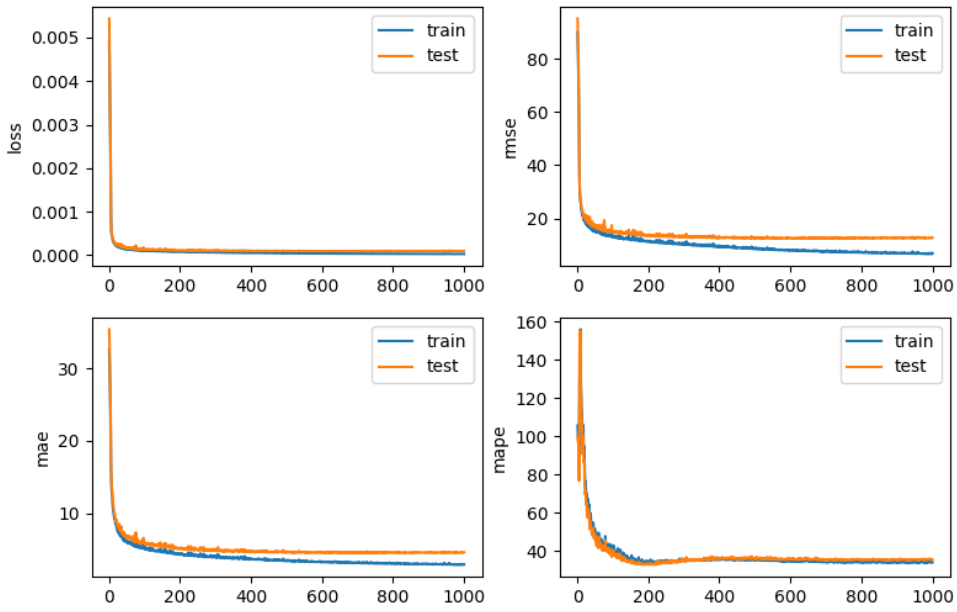

图 7 结合 3D 卷积的 LSTM 损失函数、RMSE、MAE、MAPE 变化曲线图

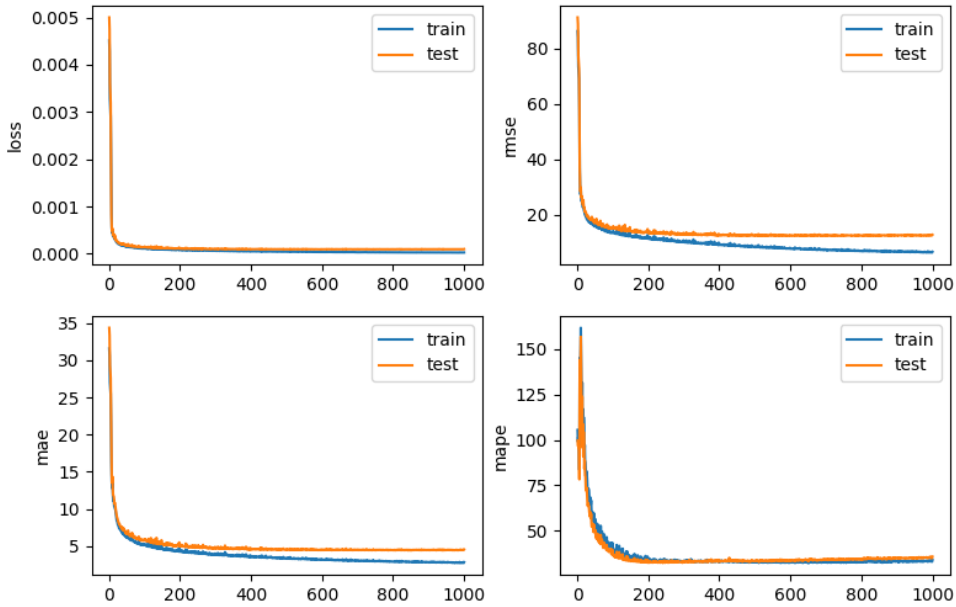

图 8 结合 3D 卷积的 LSTM 损失函数、RMSE、MAE、MAPE 变化曲线图

表 3 模型实验结果

Hidden Size	Best Epoch	RMSE	MAE	MAPE
64	548	12.38	4.54	34.00
128	696	12.33	4.48	35.17
256	627	12.23	4.46	33.58

从表 3 可以看出，隐藏层特征维度的增大导致取得最好性能需要的训练迭代次数增多，训练时间也大幅增加，但性能增益并不显著。

图 9 为 drop probability 调整为 0.3 的结合 3D 卷积的 LSTM 模型性能变化曲线图，具体的数据对比如表 4 所示。

表 4 模型实验结果

Drop Prob.	Best Epoch	RMSE	MAE	MAPE
0.3	387	12.28	4.56	32.37
0.5	548	12.38	4.54	34.00

从表 4 可以看出，drop probability 的减小，模型的复杂度获得了一定的提升，从而在一定程度上提升了模型的特征表示能力，可以更快更好地获得较好的模型性能。

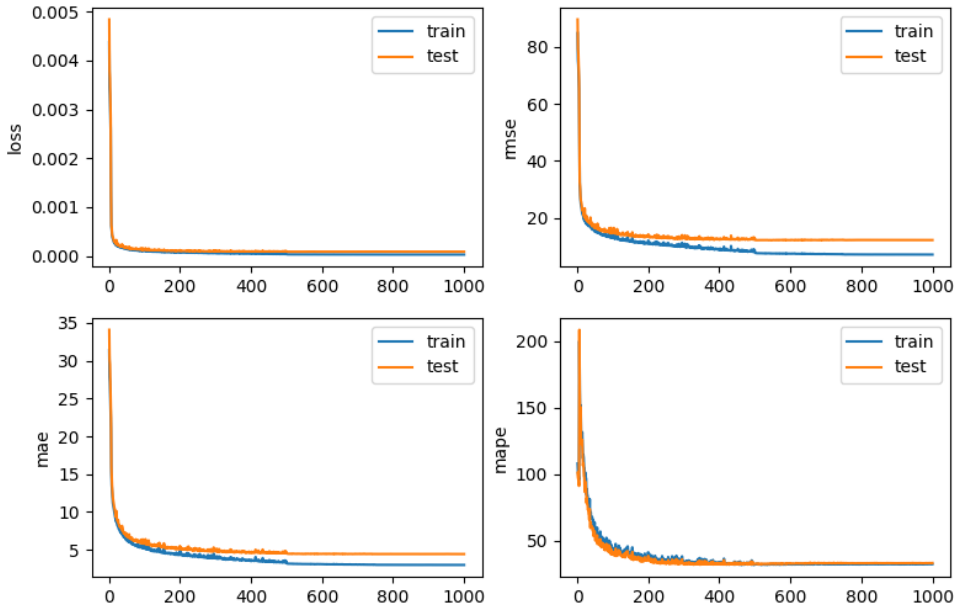

图 9 结合 3D 卷积的 LSTM 损失函数、RMSE、MAE、MAPE 变化曲线图

图 10 动态调整学习率的结合 3D 卷积的 LSTM 模型性能变化曲线图，学习率在第 500 和 725 个 Epoch 分别调整为之前的 0.1 倍，具体的数据对比如表 5 所示。

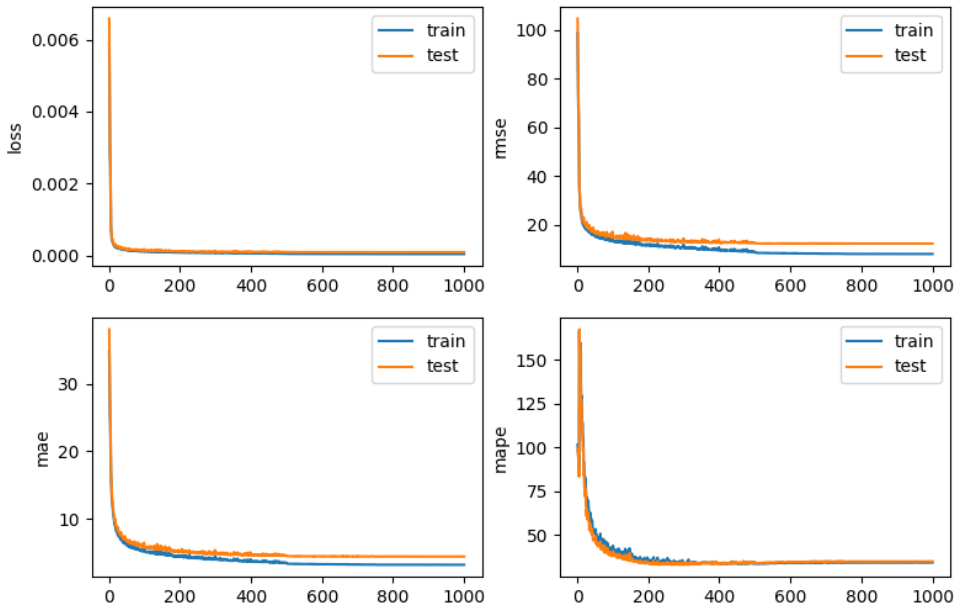

图 10 结合 3D 卷积的 LSTM 损失函数、RMSE、MAE、MAPE 变化曲线图

表 5 模型实验结果

学习率	Best Epoch	RMSE	MAE	MAPE
固定学习率	548	12.38	4.54	34.00
动态调整	665	12.26	4.47	34.84

从表 5 可以看出，由学习率设置不合适导致的震荡问题可以通过动态调整学习率获得一定的缓解，从而使得模型的训练损失和测试损失继续下降并获得性能的提升。基于上述对比实验和分析，考虑模型的性能和复杂度，最终模型的参数调整为 drop probability = 0.3，其余保持不变，并在训练过程中动态调整学习率，最终模型的性能如表 6 所示。

表 6 模型实验结果

隐藏层层数	Best Epoch	RMSE	MAE	MAPE
1	569	12.19	4.46	33.36
2	**546**	**12.14**	**4.45**	**32.95**
3	511	12.27	4.46	33.80

本文选取的顶会论文模型架构如图 11 所示，该模型在 8 张显存为 32G 的 V100 上训练，每张 GPU 上的 batch size 为 2，即总体上 batch size 为 16。本文在 4 张 V4000 上进行实验结果的复现，受显存的限制，调整每张 GPU 上的 batch size 为 1，即总体上 batch size 为 4，其余超参数保持不变。由于 YouTube VIS 2019 数据集体量较小，单独作为训练集会导致模型过拟合，一般会在 COCO 数据集上进行预训练，然后在 YouTube VIS 数据集上进行单独训练或两个数据集联合训练。本文采用了论文提供的预训练模型，在 YouTube VIS 数据集上进行单独训练和评估。实验结果如表 7 所示。

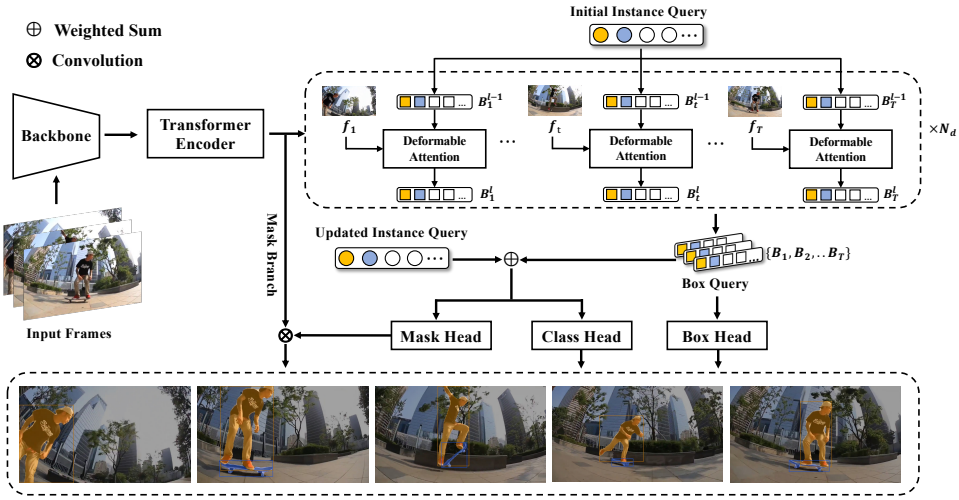

图 11 SeqFormer 模型架构示意图

表 7 SeqFormer 复现实验结果

Backbone	Method	Batch Size	Learning Rate	AP	AP_{50}	AP_{75}	AR_1	AR_{10}
ResNet-50	论文模型	16	2×10^{-4}	45.1	66.9	50.5	45.6	54.6
	复现模型	4	2×10^{-4}	41.6	62.7	45.6	42.3	50.0
	复现模型	4	5×10^{-5}	41.4	62.4	44.8	41.9	50.5

如表 7 所示，在只缩小 batch size 而不调整其他超参数（如学习率、迭代次数）的情况下，模型的性能发生了明显下降，大约 4 个点，这是由于 batch size 缩小，模型训练过程中梯度方差变大，训练不稳定，从而导致训练下降。相关研究显示在扩大 batch size 的前提下，同比例扩大学习率可以有效获得相同的性能，因此考虑同比例缩小学习率。但实验结果表明，这一策略并没有使得模型的训练更加稳定。同样的情况也出现在其他人的复现过程中。

The results is 46.96, which is lower than provided results 49.5. I'm using Torch 1.9.0, and batchsize was set to 16 instead of 32.

在将模型的 batch size 缩小为作者参数的 1/2 后，模型的性能下降了约 2 个点。为了帮助他人更好的复现论文结果，作者做出了如下实验：

- 实验一：按照原参数训练模型五次；
- 实验二：不改变训练迭代次数和学习率调整策略，将 batch size 缩小 1/2 训练模型 5 次；
- 实验三：在将 batch size 缩小为 1/2 的同时将训练迭代次数和学习率下降的 epoch 调整为原来的 2 倍，训练模型 5 次。

实验结果如表 8 所示。

表 8 SeqFormer 复现实验结果

实验	# 1 AP	# 2 AP	# 3 AP	# 4 AP	# 5 AP
实验一	50.5	49.4	49.8	49.7	48.8
实验二	47.5	47.1	47.9	47.4	48.1
实验三	49.0	47.9	48.5	49.1	48.6

除此以外，复现者也做了更多的实验来验证不同的参数调整策略对模型复现的影响。

IMS_PER_BATCH	BASE_LR	STEPS	MAX_ITER	MIN_SIZE_TEST	results
16	0.0001	(8000,)	12000	360	46.96
16	0.0001	(8000,)	**24000**	360	47.10
16	0.0001	**(16000,)**	24000	360	46.40
16	**0.0001 / 2**	(16000,)	24000	360	47.82
16	0.0001	(16000,)	24000	**480**	48.55
16	0.0001 / 2	(16000,)	24000	**480**	48.47

综合上述实验，可以得到如下结论：

- 模型的性能会有 1～1.5 个点的波动；
- 训练时 batch size 缩小 1/2，模型性能大约会下降 2 个点；
- 等比例调整学习率，并不能改变调整 batch size 对模型性能的影响，需要多个参数调整策略的综合作用才能获得相同的模型性能；
- 增大训练迭代次数和学习率调整的 epoch 可以比较有效的削弱缩小 batch size 造成的模型性能下降，但不能完全复现模型的性能。

为此，本文为了复现模型的性能，采用了梯度累加的方法来在硬件不变的前提下提升 batch size，实验结果如表 9 所示。

传统的训练函数，一个 batch 是这么训练的：

```
for i, (image, label) in enumerate(train_loader):
    # 1. input output
    pred = model(image)
    loss = criterion(pred, label)
    # 2. backward
    optimizer.zero_grad() # reset gradient
    loss.backward()
    optimizer.step()
```

1. 获取 loss：输入图像和标签，通过 infer 计算得到预测值，计算损失函数；
2. optimizer.zero_grad() 清空过往梯度；

3. loss.backward() 反向传播，计算当前梯度；

4. optimizer.step() 根据梯度更新网络参数

简单的说就是进来一个 batch 的数据，计算一次梯度，更新一次网络。使用梯度累加是这么写的：

```
for i, (image, label) in enumerate(train_loader):
    # 1. input output
    pred = model(image)
    loss = criterion(pred, label)
    # 2.1 loss regularization
    loss = loss / accumulation_steps
    # 2.2 back propagation
    loss.backward()
    # 3. update parameters of net
    if (i + 1) % accumulation_steps == 0:
        # optimizer the net
        optimizer.step() # update parameters of net
        optimizer.zero_grad() # reset gradient
```

1. 获取 loss：输入图像和标签，通过 infer 计算得到预测值，计算损失函数；

2. loss.backward() 反向传播，计算当前梯度；

3. 多次循环步骤 1-2，不清空梯度，使梯度累加在已有梯度上；

4. 梯度累加了一定次数后，先 optimizer.step() 根据累计的梯度更新网络参数，然后 optimizer.zero_grad() 清空过往梯度，为下一波梯度累加做准备。

总结来说：梯度累加就是，每次获取 1 个 batch 的数据，计算 1 次梯度，梯度不清空，不断累加，累加一定次数后，根据累加的梯度更新网络参数，然后清空梯度，进行下一次循环。

表 9 SeqFormer 复现实验结果

Backbone	Method	Batch Size	Learning Rate	AP	AP_{50}	AP_{75}	AR_1	AR_{10}
ResNet-50	论文模型	16	2×10^{-4}	45.1	66.9	50.5	45.6	54.6
	复现模型	4	2×10^{-4}	41.6	62.7	45.6	42.3	50.0
	复现模型	16	2×10^{-4}	43.8	67.4	47.5	42.1	52.4

可以看到，通过梯度累积的方式扩大 batch size，可以在模型性能波动范围内对论文进行复现，甚至在某些指标上高于论文模型的结果。

六、实验心得体会

本次实验中，综合利用了卷积神经网络和循环神经网络（或 Transformer）来解决实际问题，以取得在真实数据集上较好的模型性能。无论是出租车流量预测任务还是视频实例分割任务，都兼具序列数据的特性和局部相关性，因而可以巧妙地结合卷积神经网络捕捉局部相关性和循环神经网络（或 Transformer）捕捉序列相关性的能力，充分提取数据中的有效信息进行预测。

通过本次实验，一方面更加充分地理解了不同模型的特性和适合的使用场景，另一方面，通过自行设计模型架构并实现，理论和代码能力都有了进一步的提升，尤其是通过对顶会论文代码的阅读和复现，更加深入地掌握了 PyTorch 函数的使用方法和模型训练的调参技巧。

七、参考文献

[1] 北京交通大学 《深度学习》课程组，实验 5 综合实验（大作业），北京交通大学《深度学习》课件

[2] Junfeng Wu, Yi Jiang, Song Bai, Wenqing Zhang, and Xiang Bai. 2022. Seqformer: Sequential transformer for video instance segmentation. In European Conference on Computer Vision. 553–569.

[3] Junfeng Wu, Qihao Liu, Yi Jiang, Song Bai, Alan L. Yuille, and Xiang Bai. 2022. In Defense of Online Models for Video Instance Segmentation. In European Conference on Computer Vision. 588–605.

八、附录

无