2024《人工智能导论》大作业

任务名称：不良内容图像检测

完成组号：11

小组人员：郭嘉栋 丁智达

完成时间：2024.6.19

1．任务目标

基于给定的暴力图像数据集，实现一个用于图像分类的机器学习流程，对数据集的不良内容进行检测与识别

2．具体内容

（1）实施方案

通过采用 cnn 卷积神经网络模型，采用自定义数据集合数据加载机，利用 PyTorch 和 PyTorch Lighting 对图像中的暴力内容进行二分类进行模型训练和评估，并用所得到的模型对测试集中的元素进行判断 ，本次实验提供两个接口，一个是 classify（），另外一个是 classify1（），其中 classify 直接进行模型训练，并用训练所得到的模型对测试集进行预测。classify1 通过历史断点加载模型，并对测试集进行预测

（2）核心代码分析

下面对代码进行分析

I. cnn 基本模型框架代码

```
class Basemodel(nn.Module):
    def __init__(self, num_classes=2):
        super(Basemodel, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3,
padding=1)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3,
padding=1)
        self.conv3 = nn.Conv2d(64, 128,
kernel_size=3, padding=1)
        self.fc1 = nn.Linear(128 * 28 * 28, 512)
        self.fc2 = nn.Linear(512, num_classes)
        self.pool = nn.MaxPool2d(2, 2)
        self.dropout = nn.Dropout(0.25)
```

```
    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.pool(F.relu(self.conv3(x)))
        x = x.view(-1, 128 * 28 * 28)
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)
        return x
```

Basemodel 定义了一个卷积神经网络，并通过 PyTorch 实现了前向传播的逻辑。conv1,conv2,conv3 分别对应三个卷积层，conv1 从 3 个输入通道中提取 32 个特征图，conv2 从输入特征图中提取 64 个特征图，conv3 从 64 个输入特征图中提取 128 个特征图，每个卷积层的核大小为 3*3，填充为 1，以保持特征图大小不变。

II. 项目由两个主要的 python 文件组成

- **dataset.py 定义自定义的数据及和数据模块**
- **classifier.py 是实现神经网络架构和训练逻辑的主要部分。**

此外上传文件中还有一个 main.py 文件，主要是用来提供一个使用的范例，将在接口文件说明中展示

III.具体类作用及提供函数说明

CustomDatase 类

```
class CustomDataset(Dataset):
    def __init__(self, split):
        assert split in ["train", "val", "test"]
```

```
        data_root = "violence_224/"
        self.data = [os.path.join(data_root, split, i)
for i in os.listdir(os.path.join(data_root, split))]
        if split == "train":
            self.transforms = transforms.Compose([
                transforms.RandomHorizontalFlip(),
                transforms.ToTensor(),
            ])
        else:
            self.transforms = transforms.Compose([
                transforms.ToTensor(),
            ])

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

    def __getitem__(self, index):
        img_path = self.data[index]
        x = Image.open(img_path)
        y = int(img_path.split("\\")[-1][0])
        x = self.transforms(x)
        return x, y

    def convert_to_tensor(self):
        dataset_tensors = []
        for img_path in self.data:
            img = Image.open(img_path)
            img_tensor = self.transforms(img)
            dataset_tensors.append(img_tensor)
        return torch.stack(dataset_tensors, dim=0)
```

作用说明

这个类提供如下几个函数接口

1._init_为特定数据集分割（train,test,val）初始化数据集，它设置了数据转化，并为训练集添加了额外的数据增强

2._len_:返回数据集中样本的数量

3._getitem_：加载并返回单个样本和标签

4_convert_to_tensor:主要用于测试，将整个数据集转化为 n*3*224*224 张量

CustomDataModule 类

```
class CustomDataModule(LightningDataModule):
    def __init__(self, batch_size=32, num_workers=4):
        super().__init__()
        self.batch_size = batch_size
        self.num_workers = num_workers
    def setup(self, stage=None):
        # 分割数据集、应用变换等
       # 创建 training, validation 数据集
       self.train_dataset = CustomDataset("train")

        self.val_dataset = CustomDataset("val")

        self.test_dataset = CustomDataset("test")

    def train_dataloader(self):
        return DataLoader(self.train_dataset,
batch_size=self.batch_size, shuffle=True,
num_workers=self.num_workers, persistent_workers=True)

    def val_dataloader(self):
        return DataLoader(self.val_dataset,
batch_size=self.batch_size, shuffle=False,
num_workers=self.num_workers, persistent_workers=True)

    def test_dataloader(self):
        return DataLoader(self.test_dataset,
batch_size=self.batch_size, shuffle=False,
num_workers=self.num_workers, persistent_workers=True)
```

作用说明：

该类继承自 Pytorch Lighting 的 LightingDataModule，为数据处理提供一个清晰的接口，setup 是数据集的初始化，用于创建 test，train 和 val 数据集，而

*_dataloader 方法返分割的 dataloader 实例

ViolenceClassifier 类

```
class ViolenceClassifier(pl.LightningModule):
    def __init__(self, lr=3e-4):
        super(ViolenceClassifier, self).__init__()
        self.model = Basemodel()  # 在这里初始化模型
         self.lr = lr
        self.criterion = nn.CrossEntropyLoss()

    def forward(self, x):
        return self.model(x)

    def training_step(self, batch, batch_idx):
        inputs, labels = batch
        outputs = self(inputs)
        loss = self.criterion(outputs, labels)
        self.log('val_loss', loss)
        return loss

    def validation_step(self, batch, batch_idx):
        inputs, labels = batch
        outputs = self(inputs)
        loss = self.criterion(outputs, labels)
        self.log('val_loss', loss, prog_bar=True)
        return loss

    def configure_optimizers(self):
        optimizer = optim.Adam(self.parameters(),
lr=self.lr)
        return optimizer

    @classmethod
    def load_from_checkpoint(cls, checkpoint_path):
        # 加载模型
       model = cls()
        checkpoint = torch.load(checkpoint_path)
        model.load_state_dict(checkpoint['state_dict'])
        return model

    def test_step(self, batch, batch_idx):
        inputs, labels = batch
        outputs = self(inputs)
```

```
        loss = self.criterion(outputs, labels)

        # 计算正确率
        _, predicted = torch.max(outputs, 1)
        accuracy = (predicted == labels).sum().item() /
labels.size(0)

        # 记录正确率
        self.log('test_accuracy', accuracy, prog_bar=True)
        return loss

    def classify(self,input_tensor):
        gpu_id = [0]
        lr = 3e-4
        batch_size = 128
        log_name = "resnet18_pretrain_test"
        print("{} gpu: {}, batch size: {}, lr:
{}".format(log_name, gpu_id, batch_size, lr))
        # 创建数据模块
        data_module = CustomDataModule(batch_size=128)
        # 设置数据模块
        data_module.setup()
        # 创建模型
        model = ViolenceClassifier(lr=3e-4)
        # 实例化 ModelCheckpoint 回调以保存最佳模型
        checkpoint_callback = ModelCheckpoint(
            monitor='val_loss',
            filename=log_name + '-{epoch:02d}-
{val_loss:.2f}',
            save_top_k=1,
            mode='min',
        )
        # 实例化 TensorBoardLogger 用于记录日志
        logger = TensorBoardLogger("train_logs",
name=log_name)
        # 实例化训练器
        trainer = Trainer(
            max_epochs=100,
            accelerator='gpu',
            logger=logger,
            callbacks=[checkpoint_callback]
        )
        # 加载数据
        train_loader = data_module.train_dataloader()
```

```
        val_loader = data_module.val_dataloader()
        test_loader = data_module.test_dataloader()
        trainer.fit(model, train_loader)
        model.eval()  # 将模型设置为评估模式
       with torch.no_grad():
            outputs = model(input_tensor)
            _, predicted = torch.max(outputs, 1)
            return predicted.tolist()

    def classify1(self, input_tensor):
        ckpt_path =
"train_logs/resnet18_pretrain_test/version_0/checkpoints
/resnet18_pretrain_test-epoch=03-val_loss=0.18.ckpt"

        # 从检查点加载模型
       model =
ViolenceClassifier.load_from_checkpoint(ckpt_path)

        # 确保模型处于评估模式
       model.eval()

        # 如果有 GPU 可用，将模型和输入张量移动到 GPU
        if torch.cuda.is_available():
            model = model.cuda()
            input_tensor = input_tensor.cuda()

        # 禁用梯度计算进行推理
       with torch.no_grad():
            # 获取预测结果
          outputs = model(input_tensor)
            _, predicted = torch.max(outputs, 1)
            return predicted.tolist()
```

作用说明：

这个类继承自 **pl.LightningModule**，封装了模型的定义、训练、验证、测试和推理过程。下面对这个类中函数进行具体说明以了解这类的工作原理

def __init__(self, lr=3e-4)： lr：学习率，默认值为 3e-4，使用 Basemodel 类初始化模型。使用交叉熵损失函数。

forward (self, x)：前向传播函数，接受输入张量 x，并返回模型的输出。

training_step (self, batch, batch_idx)定义 train.fit 的训练步骤， 接受一个批次的输入 batch 和批次索引 batch_idx。并且计算输出和损失，并记录训练损失。

validation_step (self, batch, batch_idx)：类似于 training_step，但是用于验证步骤

configure_optimizers(self)：使用 Adam 优化器，并设置学习率。

load_from_checkpoint (cls, checkpoint_path)：从检查点加载模型

classify (self, input_tensor) ：外部调用接口 1，初始化并配置训练环境，包括数据模块、日志记录器和检查点回调，并且进行模型训练，并在训练结束后进行评估模式设置和推理。最后对测试集进行预测

classify 1(self, input_tensor) ：外部调用接口 2，从指定的检查点路径加载预训练模型，并进行推理。将模型和输入张量移动到 GPU（如果可用），并禁用梯度计算进行推理。

IV.关于如何实现泛化能力

数据增强：在 CustomDataModule 类中，通过对训练数据进行随机变换，增加数据的多样性。数据增强可以帮助模型学习到更加鲁棒的特征，减少对输入数据的敏感性。

Dropout 正则化：**basemodel** 在训练过程中随机丢弃神经元，减少神经网络的复杂度，防止过拟合，从而提高泛化能力。

使用交叉熵损失函数：在 **violenceclassifer** 类中使用交叉熵损失函数增强泛化能力

3．工作总结

（1）收获、心得

通过本次实验，我初步学习了 PyTorch 和 PyTorch Lightning 框架在图像分类任务中的应用，掌握了自定义数据加载器的实现方法，以及如何利用 GPU 进行模型加速训练。实际操作中，体验了完整的模型训练、验证、测试流程，对机器学习的过程有了更加深刻的理解

（2）遇到问题及解决思路

配置环境的问题：自己查相关资料解决

代码报错问题：依据提示，一步一步查找报错的原因。如报错没法正确加载数据集 violence_224/train，自己依据报错信息，调试路径和代码解决

4．课程建议

感觉整个课程太偏向于理论，没有跟实际的代码相结合，如果讲解具体机器学习的方法能够跟实际的代码相结合并且在平时提供一定的实践机会会更好