《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：王子睿 孟令泉 郝建东

完成时间：2024/6/19

1．任务目标

基于暴力图像检测数据集，构建一个检测模型。该模型可以对数据集的图像进行不良内容 检测与识别。
要求：
模型是 2 分类（0 代表正常图像、1 代表不良图像），分类准确率越高越好；
模型具有一定的泛化能力：不仅能够识别与训练集分布类似的图像，对于 AIGC 风格变 化、图像噪声、对抗样本等具有一定的鲁棒性；
有合理的运行时间。

2．具体内容

（1）实施方案

基于暴力图像检测数据集，我们可以采用以下方案：
收集数据集，并进行数据增强，如随机裁剪、旋转、颜色变换等。
选择一个深度学习模型（如 ResNet、EfficientNet），并在其基础上构建一个二分类模型。
使用交叉熵损失函数和 Adam 优化器，训练模型。
对训练好的模型进行评估，并根据需要进行模型压缩和蒸馏。

（2）核心代码分析

本实验使用 ResNet-18 模型
模型基本框架：model.py，dataset.py 基本采用示例代码形式，具体分析如下

```python
class ViolenceClassifier(LightningModule):
    def __init__(self, num_classes=2, learning_rate=1e-3):
        super().__init__()
        self.model = models.resnet18(pretrained=True)
        num_ftrs = self.model.fc.in_features
        self.model.fc = nn.Linear(num_ftrs, num_classes)
        # self.model = models.resnet18(pretrained=False, num_classes=2)

        self.learning_rate = learning_rate
        self.loss_fn = nn.CrossEntropyLoss()  # 交叉熵损失
        self.accuracy = Accuracy(num_classes=2)

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

    def configure_optimizers(self):
        optimizer = torch.optim.Adam(self.parameters(), lr=self.learning_rate)  # 定义优化器
        return optimizer

    def training_step(self, batch, batch_idx):
        x, y = batch
        logits = self(x)
        loss = self.loss_fn(logits, y)
        self.log('train_loss', loss)
        return loss

    def validation_step(self, batch, batch_idx):
        x, y = batch
        logits = self(x)
        loss = self.loss_fn(logits, y)
        preds = torch.argmax(logits, dim=1)
        acc = self.accuracy(preds,y)
        self.log('val_loss', loss)
        self.log('val_acc', acc)
        return loss

    def test_step(self, batch, batch_idx):
        x, y = batch
        logits = self(x)
        loss = self.loss_fn(logits, y)
        preds = torch.argmax(logits, dim=1)
        acc = self.accuracy(preds,y)
        self.log('test_acc', acc)
        return loss
```

上述为重要类 ViolenceClassifier 的定义，其定义于 model.py 中，
它具体实现了：
（1 初始化方法 (__init__)：
使用 super().__init__()调用父类 LightningModule 的初始化方法。
加载预训练的 ResNet-18 模型。
定义了学习率 learning_rate。
使用了交叉熵损失函数 nn.CrossEntropyLoss()。
使用了 torchmetrics 的 Accuracy 库来计算模型的准确度。
（2 前向传播方法 (forward)：
定义了模型的前向传播过程，即输入 x 经过模型后的输出。
（3 配置优化器方法 (configure_optimizers)：
定义了模型的优化器- Adam 优化器，并将模型的所有参数作为优化器的参数，
学习率为 self.learning_rate。
（4 训练步骤方法 (training_step)：
在每个训练批次上执行的方法。
从批次中获取输入 x 和标签 y。
计算模型的输出（logits）。
计算损失。
使用 self.log 记录训练损失，以便在训练过程中进行监控。
返回损失值，PyTorch Lightning 将自动进行反向传播和优化。
（5 验证步骤方法 (validation_step)：

在每个验证批次上执行的方法，与 training_step 类似，但还计算了预测的准确度并记录了验证损失和准确度。
（6 测试步骤方法（test_step）:
在每个测试批次上执行的方法，与 validation_step 类似，但只记录了测试的准确度。

```python
class CustomDataset(Dataset):
    def __init__(self, split):
        assert split in ["train", "val", "test"]
        data_root = r".\12-others\violence_224"
        split_dir = os.path.join(data_root, split)
        self.data = [os.path.join(split_dir, i) for i in os.listdir(split_dir)]
        if not self.data:
            raise ValueError(f"The {split} dataset is empty. Please check the {split_dir} directory.")

        if split == "train":
            self.transforms = transforms.Compose([
                transforms.RandomHorizontalFlip(),  # 随机翻转
                transforms.ToTensor(),  # 将图像转换为Tensor
            ])
        else:
            self.transforms = transforms.Compose([
                transforms.ToTensor(),  # 将图像转换为Tensor
            ])

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

    def __getitem__(self, index):
        img_path = self.data[index]
        # 使用os.path.basename获取文件名，并使用splitext分割文件名和扩展名
        filename_with_ext = os.path.basename(img_path)
        filename, file_extension = os.path.splitext(filename_with_ext)
        # 获取标签值，0代表非暴力，1代表暴力
        # 假设文件名的主体（不包括扩展名）的第一个字符是标签
        y = int(filename[0])

        # 验证标签是否为0或1
        if y not in [0, 1]:
            raise ValueError(f"Invalid label: {y} in file {img_path}")

        x = Image.open(img_path)
        x = self.transforms(x)
        return x, y
```

以上是定义于 dataset.py 中的重要类 CustomDataset，它继承自 PyTorch 的 Dataset 类，用于加载自定义的数据集。
其实现了__getitem__ 方法，数据集类中最关键的方法，允许通过索引访问数据集中的样本。
首先，通过索引从 self.data 中获取图像文件的路径。使用 os.path.basename 获取文件名（不包括路径），并使用 os.path.splitext 分割文件名和扩展名。文件名的主体（不包括扩展名）的第一个字符是标签（0 代表非暴力，1 代表暴力）。对获取的标签进行验证，确保它是 0 或 1。如果不是，则抛出一个 ValueError 异常。使用 PIL 的 Image.open 打开图像文件，并应用之前定义的 transforms（数据预处理）。最后，返回预处理后的图像（x）和对应的标签（y）。

```python
class otherDataset(Dataset):
    def __init__(self, path) -> None:
        super().__init__()
        self.data = [os.path.join(path, i) for i in os.listdir(path)]

        self.t = transforms.Compose([
            transforms.Resize((224, 224)),
            transforms.ToTensor()
        ])

    def __getitem__(self, index):
        img_path = self.data[index]
        # 使用os.path.basename获取文件名，并使用splitext分割文件名和扩展名
        filename_with_ext = os.path.basename(img_path)
        filename, file_extension = os.path.splitext(filename_with_ext)
        # 获取标签值，0代表非暴力，1代表暴力
        # 假设文件名的主体（不包括扩展名）的第一个字符是标签
        y = int(filename[0])

        # 验证标签是否为0或1
        if y not in [0, 1]:
            raise ValueError(f"Invalid label: {y} in file {img_path}")

        x = Image.open(img_path)
        x = self.t(x)
        return x, y

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

以上是在 dataset.py 中增加的 otherDataset 类，与原先定义的 CustomDataset 类似，这个类主要用于 classify.py 中检测不同的数据集得到结果

```python
from pytorch_lightning import Trainer
from pytorch_lightning.callbacks import ModelCheckpoint
from pytorch_lightning.loggers import TensorBoardLogger

from model import ViolenceClassifier
from dataset import CustomDataModule

if __name__ == "__main__":
 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))

 data_module = CustomDataModule(batch_size=batch_size)
 # 设置模型检查点，用于保存最佳模型
 checkpoint_callback = ModelCheckpoint(
    monitor='val_loss',
    filename=log_name + '-{epoch:02d}-{val_loss:.2f}',
    save_top_k=1,
    mode='min',
 )
 logger = TensorBoardLogger("train_logs", name=log_name)

 # 实例化训练器
 trainer = Trainer(
    max_epochs=40,
    accelerator='gpu',
    devices=gpu_id,
    logger=logger,
    callbacks=[checkpoint_callback]
 )

 # 实例化模型
 model = ViolenceClassifier(learning_rate=lr)
 # 开始训练
 trainer.fit(model, data_module)
```

以上为 train.py 代码截图，在其中定义了使用 GPU ID,批量大小，学习率等等，并初始化数据模块，设置模型检查点，初始化 TensorBoard 日志记录器，初始化模型，并开始训练。得到的训练日志及文件在目录 train_logs\resnet18_pretrain_test 下，本实验最终使用 version_7 得到的模型。

剩余可分析的代码还有生成噪声图片的 data_gen.py,及接口类 classify.py，前者在整个体系中显得没那么重要，而后者会由 markdown 文件具体解释，在此不加赘述。

测试结果：

```
(test) E:\test>python 12-classify.py
test1 acc:0.987353206865402, test2 acc:0.875, test3 acc:0.9

(test) E:\test>
```


3. 工作总结

（1）收获、心得

完成这个基于暴力图像检测的数据集模型构建项目，我将收获以下几点：

技术提升：我将深入理解深度学习在图像分类任务中的应用，如卷积神经网络（CNN）特别是针对图像识别的 ResNet、Inception 等架构，以及迁移学习（如使用预训练的 VGG16 或 EfficientNet）

数据处理：我将学会如何对原始数据进行预处理，包括图像增强（如旋转、缩放、裁剪）、归一化，以及如何处理 AIGC 风格变化、噪声和对抗样本。

模型调优：通过交叉验证和超参数调整，我将提高模型的准确率和泛化能力，理解如何平衡精度和运行时间。

鲁棒性理解：通过对抗训练和模型鲁棒性测试，我将理解模型在面对不同输入变化时的性能，这将增强模型的实用性。

实践经验：实际操作中，我将学会如何在实际环境中部署和测试模型，包括性能监控和实时响应。

心得总结：这个项目锻炼了我的编程能力，加深了对深度学习原理的实践应用，同时让我认识到在实际场景中模型的复杂性和挑战。通过解决这些问题，我不仅提升了技术能力，也学会了如何在不断变化的数据和需求中迭代优化模型。

（2）遇到问题及解决思路

过拟合：模型可能在训练集上表现很好，但在未见过的数据上表现不佳。

解决思路：增加正则化（如 L1/L2 正则化），使用 Dropout 技术，或者增加数据集的多样性。

泛化能力差：模型可能无法有效识别风格变化、噪声或对抗样本。

解决思路：使用数据增强技术模拟不同的图像变化，进行对抗训练以增强模型对对抗样本的鲁棒性。

运行时间过长：模型可能在实际应用中运行效率低下。

解决思路：优化模型结构，如使用轻量级网络（如 MobileNet），或者使用模型剪枝和量化技术减少模型大小和计算量。

模型解释性差：深度学习模型通常被认为是“黑箱”，难以解释其决策过程。

解决思路：使用可视化技术（如 Grad-CAM）来理解模型如何做出决策，或者使用可解释性强的模型（如决策树）作为辅助。

4．课程建议

课程中应包含足够的实践环节，如编程作业、项目实践和实验室工作，以加深学生对理论知识的理解。同时可以确保学生掌握必要的数学和编程基础，为深入学习复杂的人工智能算法打下坚实基础。

其次，随着人工智能领域的快速发展，课程内容应定期更新，以反映最新的技术和研究进展。可以结合当前人工智能领域的热点案例，如 ChatGPT、自动驾驶等，让学生了解 AI 技术的实际应用和挑战。

最后，鼓励学生提出创新性的想法，可以从这些想法中选一些可实现性强的。设计综合性项目，让学生从问题定义、数据收集到模型训练和评估的整个流程中，体验人工智能项目的完整生命周期，同时培养学生的创新能力和问题解决能力。