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：2

小组人员：杜宇欣 马思远 王亿 何润南

完成时间：2024 年 6 月

1．任务目标

基于提供的暴力图像检测数据集，一个具备一定的鲁棒性和泛化能力、具备合理运行时间的检测模型，实现对暴力与非暴力图像的高精度二分类。

2．具体内容

（1）实施方案

① 在老师给出的参考代码的基础上，通过使用不同的训练模型（resnet18、resnet50 等），修改超参数（训练轮次、学习率等）进行多次训练，对比训练结果在训练集同源数据集，AIGC 图像集和加上噪声后的图像集上的正确率，全面评估模型在不同场景下的性能表现，选择效果最好的模型进行提交。
② 杜宇欣、马思远负责接口文件的编写、模型训练以及报告撰写；王亿负责噪声添加，何润南负责 AIGC 生成图像。

（2）核心代码分析

① 环境依赖

实验中使用的外部库主要有：导入 torch 库用以提供深度学习框架，并且使用其中的 Dataset 类、DataLoader 类等进行后续操作等；使用 pytorch_lightning 中的 LightningDataModule 类简化训练过程，使用其中的 Trainer 类进行模型的训练和测试等；利用 PIL 库对图像数据集进行处理；利用 numpy 库进行数组有关操作等，训练或测试时也调用了一些类中定义好的方法。代码各部分根据需要要导入不同的库，下面不再一一列举。
② 数据加载部分

```python
class CustomDataset(Dataset):
    def __init__(self, split):
        assert split in ["train", "val", "test"]
        data_root = "C:\\Users\\86157\\Desktop\\violence_224\\"
        self.data = [os.path.join(data_root, split, i) for i in os.listdir(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

class CustomDataModule(LightningDataModule):
    def __init__(self, batch_size=32, num_workers=0):
        super().__init__()
        self.batch_size = batch_size
        self.num_workers = num_workers

    def setup(self, stage=None):
        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)

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

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

代码主要定义了两个类：CustomDataset 类和 CustomDataModule 类。

CustomDataset 类继承自 PyTorch 的自定义数据集 Dataset 类，主要实现数据的加载和预处理。其初始化函数接受 split 参数（参数的选择为 train、val 、或 test ），并且指定数据集的根目录 data_root，“self.data = [os.path.join(data_root, split, i) for i in os.listdir(data_root + split)]”语句会把根路径下的所有文件路径存储在列表 self.data 中。初始化函数会根据 split 参数对数据集做不同处理，val 和 test 对应的数据集会仅将图像转换为 Tensor 格式，而 train 数据集还会随机翻转图像，以增强模型的鲁棒性。__getitem__ 函数会根据索引参数，通过 int(img_path.split("/")[-1][0])获取图像标签供训练使用，还会对图像进行预定义的处理，最终返回处理过的图像与标签。

CustomDataModule 类继承自 PyTorch Lightning 的 LightningDataModule 类，后者通过将数据集的加载、分割和预处理步骤集中在一个类中简化了数据加载和代码结构。该类主要实现数据集的建立和加载器的创建。初始化函数主要设置批次大小 batch_size 和工作进程数 num_workers，setup 函数将数据模块分割为训练集、验证集、测试集，并通过 CustomDataset("train")等调用构造 CustomDataset 对象，将训练集等实例化。下面定义的 train_dataloader 函数调用 Pytorch 中的 DataLoader 类，按批次加载处理数据集。

③ 模型创建部分

```python
class ViolenceClassifier(LightningModule):
    def __init__(self, num_classes=2, learning_rate=1e-3):
        super().__init__()
        self.model = models.resnet50(pretrained=True)
        num_ftrs = self.model.fc.in_features
        self.model.fc = nn.Linear(num_ftrs, num_classes)

        self.learning_rate = learning_rate
        self.loss_fn = nn.CrossEntropyLoss()
        self.accuracy = Accuracy(task="multiclass", 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)
        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)
        acc = self.accuracy(logits, 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)
        acc = self.accuracy(logits, y)
        self.log('test_acc', acc)
        return acc
```

代码主要定义了一个 ViolenceClassifier 类，该类是一个继承自

LightningModule 的自定义模型类，即为训练的模型主体。类中定义了六个函数。

初始化函数接受参数 num_classes 作为输出类别数，默认为 2；参数 learning_rate 作为学习率，默认为 0.001；同时通过 self.model = models.resnet18(pretrained=True)，加载预训练的 ResNet18 模型（ResNet-18 是一种深度卷积神经网络，此处可以使用不同的模型，如更改为 Resnet50 模型）。初始化函数中还设置了交叉熵损失函数 self.loss_fn 为 nn.CrossEntropyLoss()、定义准确率指标等。

forward 函数主要是定义前向传播逻辑，接收输入并返回模型的输出；configure_optimizers 函数根据学习率等定义要采用的优化器 optimizer，此处使用的为 adam 优化器；training_step 函数是定义了一步训练操作。具体操作有获取批次输入 batch、通过 logits = self(x)前向传播计算结果、计算并返回训练损失 loss 等。validation_step、test_step 函数作用类似，操作过程也基本相同；验证步骤会另外计算验证准确率 acc，测试步骤则直接计算返回准确率。

④ 模型训练部分

```
gpu_id = [1]
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)
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)

trainer = Trainer(
    max_epochs=40,
    accelerator='gpu',
    devices=gpu_id,
    logger=logger,
    callbacks=[checkpoint_callback]
)
model = ViolenceClassifier(learning_rate=lr)
trainer.fit(model, data_module)
```

这部分代码首先指定并输出了使用 GPU 的 id、学习率 lr、批量大小 batch_size 等参数；然后实例化一个 CustomDataModule()类的对象作为数据模块，并传入批次大小。

训练时设置了检查点 checkpoint_callback，可以监控验证损失 val_loss 并将损失最小的模型进行保存；然后进行 Trainer 对象的实例化，设置最大训练轮数、使用 gpu 加速等。然后实例化 ViolenceClassifier 类创建 model 对象，设置学习率。最后调用 Trainer 类中的 fit 方法，传入模型和数据模块，开始训练。

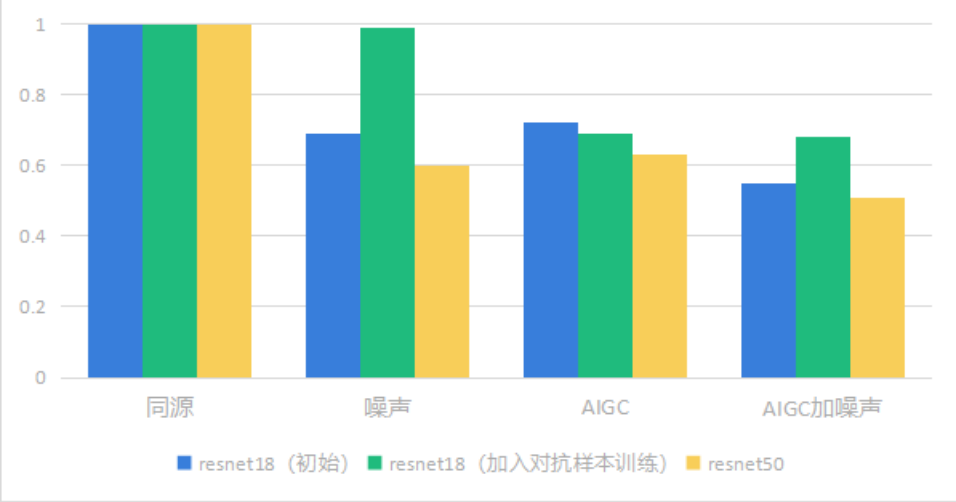
⑤ 模型测试部分

```
gpu_id = [0]
batch_size = 128
log_name = "resnet18_pretrain"

data_module = CustomDataModule(batch_size=batch_size)
ckpt_root = "C:\\Users\\86157\\Desktop\\人工智能导论\\train_logs\\"
ckpt_path = ckpt_root + "resnet18_pretrain_test/version_5/checkpoints/resnet18_pretrain_test-epoch=38-val_loss=0.04.ckpt"
logger = TensorBoardLogger("test_logs", name=log_name)

model = ViolenceClassifier.load_from_checkpoint(ckpt_path)
trainer = Trainer(accelerator='gpu', devices=gpu_id)
trainer.test(model, data_module)
```

这部分供测试使用。首先也是配置 gpu 的 id、训练批次等参数，然后指定模型检查点的路径，并且从检查点加载模型；再实例化 trainer 对象，调用 Trainer 类中的 test 方法，传入模型和数据模块进行测试。

（3）测试结果

下图为初始模型、使用 resnet50 训练的模型、加入噪声进行对抗训练的模型在不同测试集上的准确率：



模型	同源	噪声	AIGC	AIGC加噪声
resnet18	1	0.69	0.72	0.55
resnet18	1	1	0.69	0.68
resnet50	1	0.6	0.63	0.51

实验中采用的噪声为高斯噪声，直接训练的模型抗噪声能力较弱，对 aigc 图像准确率一般；resnet50 模型则仅在同源数据集上表现较好，怀疑是过拟合所致（为防止污染没有在训练集中加入 AIGC 图像）；加入噪声样本对抗训练的最终模型抗噪声能力有明显提升，但是对 AIGC 图像的判断率没有明显变化。

3. 工作总结

（1）收获心得

经过本次实验，我们不仅进一步熟悉了训练人工智能模型的过程，尤其是深刻地认识到了深度学习作为辅助人类的强大工具所具备的巨大潜力。同时，我们也初步掌握了 Python 相关工具的使用，对图像处理有了更具体的了解，大大提升了动手实践能力。

（2）遇到问题及解决思路

①. 使用 resnet50 模型准确率不高：怀疑原因是过拟合，降低训练轮次后有所改善，但效果仍不理想。

②. 通过接口文件加载模型时不匹配：classify.py 文件中忘记设置输出类别数，设置为 2 后问题解决。

③. GPU 不支持推荐版本的 pytorch，运行速度很慢：通过更换训练设备解决了该问题。

4．课程建议

①.在讲理论知识以外能多讲一些代码实现；
②.大作业最好早一点布置，太晚布置会和期末复习重合，时间比较赶。