暴力图像检测项目

项目概述

本项目实现了一个用于图像分类的机器学习流程，专门设计用于检测图像中的暴力内容。项目使用PyTorch和PyTorch Lightning进行模型训练和评估，并采用自定义数据集和数据加载机制。使用的模型是卷积神经网络

使用模型说明

以下是基本模型的代码和解释说明

```
def __init__(self, num_classes=2):#初始化
    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):#向前推进
    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
```

1. conv1, conv2, conv3: 三个卷积层。conv1从3个输入通道提取32个特征图，conv2从32个输入特征图提取64个特征图，conv3从64个输入特征图提取128个特征图。每个卷积层的核大小为3x3，填充为1，以保持特征图的大小不变。

2. fc1, fc2: 两个全连接层。fc1接收来自卷积层输出的128 * 28 * 28维向量，并将其映射到512维。fc2将512维的特征映射到类别数量（默认为2）。

3. pool: 最大池化层。每次操作从特征图中提取最大值，通过2x2窗口和2步长来减小特征图的空间尺寸。

4. dropout: 丢弃层。以0.25的概率随机丢弃神经元，用于减少过拟合。

5. forward 方法定义了网络的前向传播过程，即输入数据如何通过网络层传递。

6. 通过 conv1, conv2, conv3 进行卷积操作，接着通过 ReLU 激活函数进行非线性变换，并通过 pool 最大池化层进行空间降采样。

7. x.view(-1, 128 * 28 * 28) 将特征张量展平为一维，以便送入全连接层 fc1。

8. fc1 通过 ReLU 激活函数，然后通过 dropout 层进行随机丢弃操作，以减少过拟合。最后，通过 fc2 输出最终的分类结果。

文件介绍

项目由两个主要的Python文件组成：

1. `dataset.py`: 定义自定义数据集和数据模块。
2. `classifier.py`: 是实现神经网络架构和训练逻辑的主要部分包含主要的训练脚本,处理模型测试和评估，并且提供相关的接口文件
3. `main.py`：主要是为了提供一个使用的范例

具体分析

1. dataset.py

该文件定义了项目的数据处理组件。

主要类：

CustomDataset类

```
class CustomDataset(Dataset):
    def __init__(self, split):
        # ...

    def __len__(self):
        # ...

    def __getitem__(self, index):
        # ...

    def convert_to_tensor(self):
        # ...
```

- `__init__`: 为特定的数据集分割（训练、验证或测试）初始化数据集。它设置了数据转换，并为训练集添加了额外的数据增强。
- `__len__`: 返回数据集中样本的数量。
- `__getitem__`: 加载并返回单个样本及其标签。
- `convert_to_tensor`: 将整个数据集转换为n* 3 * 224 * 224张量

CustomDataModule类

```
class CustomDataModule(LightningDataModule):
    def __init__(self, batch_size=32, num_workers=4):
        # ...

    def setup(self, stage=None):
```

```
        # ...

    def train_dataloader(self):
        # ...

    def val_dataloader(self):
        # ...

    def test_dataloader(self):
        # ...
```

- 继承自PyTorch Lightning的`LightningDataModule`，为数据处理提供了一个清晰的接口。
- `setup`: 创建训练、验证和测试数据集。
- `*_dataloader`方法: 返回各个分割的DataLoader实例，配置了适当的参数。

2. classifier.py

该文件定义了神经网络架构和训练逻辑。

主要组件:

Basemodel类

```
class Basemodel(nn.Module):
    def __init__(self, num_classes=2):
        # ...

    def forward(self, x):
        # ...
```

- 定义了一个卷积神经网络，包含三个卷积层和两个全连接层。
- `forward`: 实现网络的前向传播。

ViolenceClassifier类

```
class ViolenceClassifier(pl.LightningModule):
    def __init__(self, lr=3e-4):
        # ...

    def forward(self, x):
        # ...

    def training_step(self, batch, batch_idx):
        # ...

    def validation_step(self, batch, batch_idx):
        # ...
```

```
    def configure_optimizers(self):
        # ...

    @classmethod
    def load_from_checkpoint(cls, checkpoint_path):
        # ...

    def test_step(self, batch, batch_idx):
        # ...

    def classify(self, batch_tensor):
        # ...

    def train_model(self)
        # ...

    def test(self)
        # ...
```

- 继承自PyTorch Lightning的`LightningModule`，封装了模型架构和训练逻辑。
- `training_step`, `validation_step`, `test_step`: 定义了training，validation，test的逻辑。
- `configure_optimizers`: 设置Adam优化器。
- `load_from_checkpoint`: 类方法，用于从检查点加载训练好的模型。
- `predict_batch`: 用于对一批数据进行预测的方法。
- `train`设置数据模块、模型、回调和日志记录器\初始化PyTorch Lightning的Trainer、开始训练过程并对测试集进行预测
- `test` 主要目的是加载预训练的模型并在测试集上进行评估，同时记录和保存相关的训练日志和模型检查点。最后返回的模型也可以用于测试

3. main.py

提供了一个使用示例，详细可看接口文件使用说明文档