report

张羽仪、徐菲悦、彭娜、花艺

June 2024

1 任务目标

1. 训练并实现一个暴力图像检测模型，具备高准确率的二分类能力。
2. 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，还对 AIGC 风格变化、图像噪声、对抗样本等具有一定的鲁棒性。
3. 确保模型的运行时间合理，能够在实际应用中高效运行。

2 具体内容

2.1 目的分析

本任务的目标是实现一个暴力检测模型，需要对暴力这一复杂的语义概念提取并分析其特征。通过观察原始图片，可以发现暴力主要表现为打架和冲突等行为。因此，在进行模型检测时，需要重点关注这些特征，并结合场景中的其他相关元素，如肢体动作、表情、以及周围环境等，来全面识别和判断暴力行为。

2.2 实施方案

2.2.1 实验步骤

1. 初始训练:
   - 在原有的数据集上分别训练模型。
   - 测试每个模型在 AIGC、对抗样本和同源数据集上的准确率。

2. 数据增强与再训练:

   - 在训练集中加入对抗样本、加噪数据和 AIGC 数据。
   - 再次训练每个模型，并再次测试其在不同 AIGC、对抗样本和同源数据集上的表现。

3. 反复测试和优化:

   - 根据测试结果，不断优化数据增强和训练策略。
   - 反复进行训练和测试，逐步提高模型的准确率和鲁棒性。

2.2.2 数据集的创建

为了使模型具有足够的泛化能力，用于训练和测试的数据集由自然图像、AIGC 图像、对抗样本图像、加噪图像组成。为了与模型中的卷积层相对应，图像的分辨率均为 224×224。

1. 自然图像:

   - 使用教师所提供的“violence-224”数据集。
   - 网站搜到的同源暴力打架数据集和普通图像数据集

2. AIGC 图像:

   - 使用 Stable Diffusion AIGC 模型。
   - 为了使 AIGC 的风格足够泛化,选用了 ChilloutMix-ni-fp16、Realistic-Vision-V6.0-B1、AniThing-V2.0-Pruned、AniVerse-V4.0-Pruned 四个不同风格的 Checkpoints 模型进行图像的生成。
   - 每种模型下有 200-300 张图像，其中暴力与非暴力图像的比例约为 1:1，保证了数据集的平衡性。

3. 对抗样本图像:

   - 使用训练过的 resnet18 模型的权重。
   - 将“violence-224”自然图像输入模型，根据图像的梯度，利用 FGSM 算法，在原图上加入肉眼难以辨认的细微扰动，得到对抗样本图像。

4. 加噪图像：

- 通过 python 程序，为 “violence-224” 数据集中的图像随机添加泊松噪声或椒盐噪声，比例为 1：1。
- 高斯噪声参数设置：高斯分布均值为 0，方差为 0.3。
- 椒盐噪声参数设置：添加椒噪声概率为 0.02，即各个图像中有 2% 的像素被替换为白色（255）。添加盐噪声概率为 0.02，即各个图像中有 2% 的像素被替换为黑色（0）。

图 1 到图 8 是以上数据集的示例。

2.2.3 模型的选择

我们从 $resnet18$, $resnet34$, $resnet50$, $efficientb0$, $efficientb3$, vit, CGG, $alexnet$ 等多个模型中比较选择。由于在同源数据集上各个模型都较高，因此只考虑比较 AIGC 测试集正确率、对抗样本正确率、加噪样本正确率的指标，不同模型间的比较如表 1 所示。

表 1: 模型的比较与选择

模型	模型可训练参数量	AIGC 测试集正确率	对抗样本正确率	加噪样本正确率
vit	85.5M	58.55%	-	52.33%
resnet18	11.2M	71.26%	94.21%	68.53%
resnet34	22.4M	88.97%	94.73%	91.01%
resnet50	23.5M	83.18%	93.06%	94.44%
efficientb0	4.2M	61.66%	-	75.33%
efficientb3	10.6M	69.66%	-	76.68%

综合考量，模型 $resnet34$ 的效果较好。其他模型表现欠佳的原因可能是 $ResNet18$, $EfficientNetB0$ 属于较浅的残差网络，具有较少的参数和较低的计算复杂度，导致模型容易不鲁棒或过拟合。而 ViT, $ResNet50$ 擅长处理大型数据集和复杂的视觉任务，具有较强的全局特征捕捉能力，但参数量过大，训练难度较高。

发现在训练中，模型参数量较大的模型通常难以训练，但它们在对抗样本和加噪样本方面具有较高的鲁棒性；相反，模型参数量较小的模型容易训练，但在应对复杂数据集和挑战性任务时可能表现不佳。

图 1: 对抗样本（1）

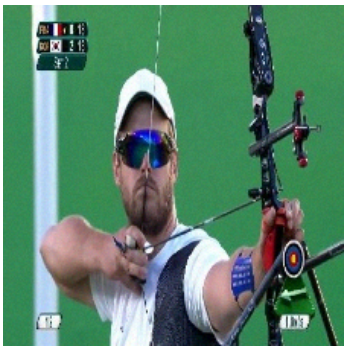

图 2: 对抗样本（2）

图 3: 同源样本（1）

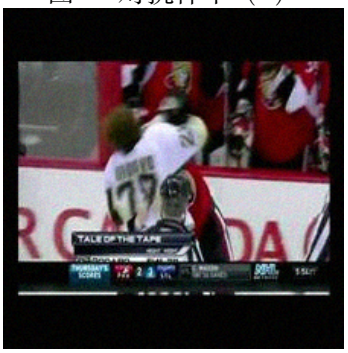

图 4: 同源样本（2）

图 5: 高斯噪声样本（1）

图 6: 高斯噪声样本（2）

图 7: 椒盐噪声样本（1）

图 8: 椒盐噪声样本（2）

2.3 核心代码分析

2.3.1 dataset.py

为了增强模型的鲁棒性，我们对训练集和验证集分别应用不同的预处理和增强策略。具体代码详见 Listing 1。

数据增强:

1. 对于训练集:
    - 调整大小
    - 随机旋转
    - 随机水平翻转
    - 随机更改图像的亮度、对比度、饱和度和色调
    - 转换为张量和归一化
2. 对于验证集:
    - 调整大小和转换为张量

```
if split == "train":
    # 定义训练集的数据预处理步骤
    self.transforms = transforms.Compose([
        transforms.Resize([224, 224]),  # 调整图像大小为224x224
        transforms.RandomRotation(45),  # 随机旋转图像，旋转角度在-45到45度之间
        transforms.RandomHorizontalFlip(),  # 随机水平翻转图像
        transforms.ColorJitter(),  # 随机更改图像的亮度、对比度、饱和度和色调
        transforms.ToTensor(),  # 将图像转换为Tensor
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])  # 标准化图像，使用ImageNet的均值和标准差
    ])
else:
    # 定义验证集或测试集的数据预处理步骤
    self.transforms = transforms.Compose([
        transforms.Resize([224, 224]),  # 调整图像大小为224x224
        transforms.ToTensor(),  # 将图像转换为Tensor
```

```
        transforms.Normalize([0.485, 0.456, 0.406], [0.229,
            0.224, 0.225])  # 标准化图像，使用ImageNet的均值和
            标准差
    ])
```

Listing 1: 训练集和验证集的数据预处理

2.3.2 model.py

```
class Resnet34ViolenceClassifier(LightningModule):
    def __init__(self, num_classes=2, learning_rate=1e-3):
        super().__init__()
        self.model = models.resnet34(pretrained=True)
        num_ftrs = self.model.fc.in_features
        self.model.fc = nn.Linear(num_ftrs, num_classes)
        # self.model = models.resnet34(pretrained=False,
            num_classes=2)

        self.learning_rate = learning_rate
        self.loss_fn = nn.CrossEntropyLoss() # 交叉熵损失
        self.accuracy = Accuracy(task="multiclass", num_classes
            =2)
```

Listing 2: ResNet34 模型训练

模型训练使用预训练的 ResNet34（核心代码详见 Listing 2），可以加快训练收敛速度，并利用预训练模型的特征提取能力，提高模型性能。通过继承 LightningModule 简化训练和验证流程，避免重复的训练代码，使代码更加清晰和模块化。

模型结构调整： 通过替换 ResNet34 的全连接层，适应特定的二分类任务。

评价指标： 使用 torchmetrics 库中的 Accuracy 类来计算多分类任务的准确率，方便简洁。

自动化日志记录： 使用 self.log 函数记录训练和验证过程中的损失和准确率，可以方便地集成到可视化工具（如 TensorBoard）中进行监控。

2.3.3 train.py

在 train.py 中设置训练参数和模型检查点，实现训练器和模型的实例化后开始训练，其核心流程如图 9 所示。

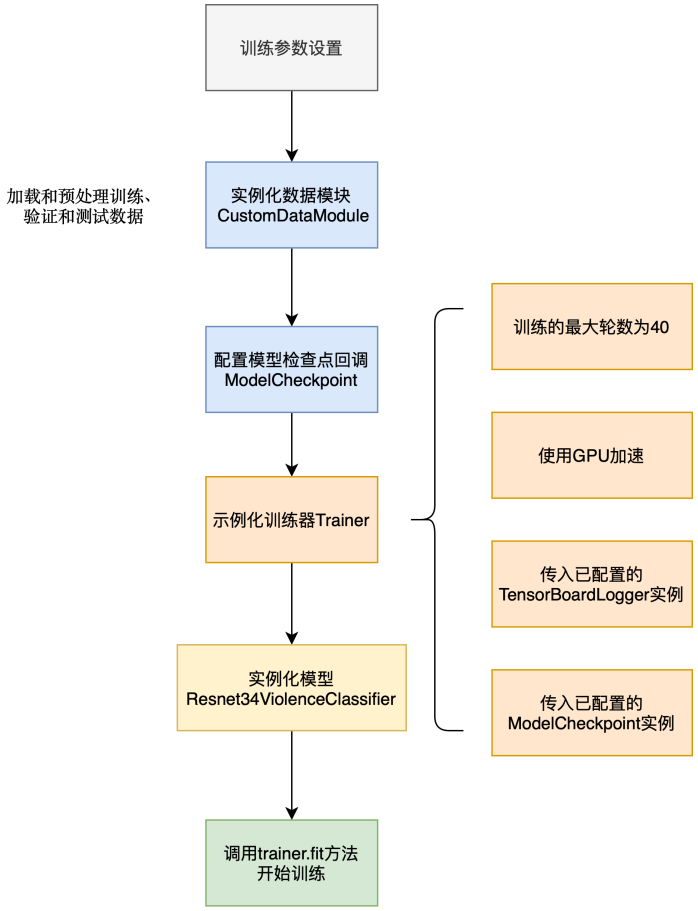

图 9: train.py 的流程图

checkpoint 保存：设置模型检查点，用于保存最佳模型，代码详见 Listing 3。

```
checkpoint_callback = ModelCheckpoint(
    monitor='val_loss',
    filename=log_name + '-{epoch:02d}-{val_loss:.2f}',
    save_top_k=5, #1,
    mode='min',
```

```
)
```

Listing 3: checkpoint 保存

实例化：训练器和模型实例化，代码详见 Listing 4。

```python
# 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 = Resnet34ViolenceClassifier(learning_rate=lr)

# 开始训练
trainer.fit(model, data_module)
```

Listing 4: 训练器和模型实例化

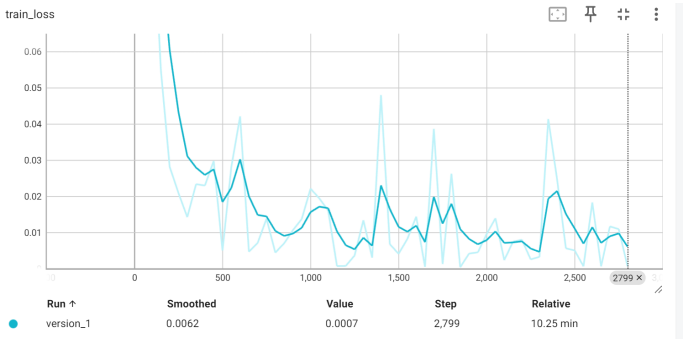

图 10: 训练过程 train loss 的曲线

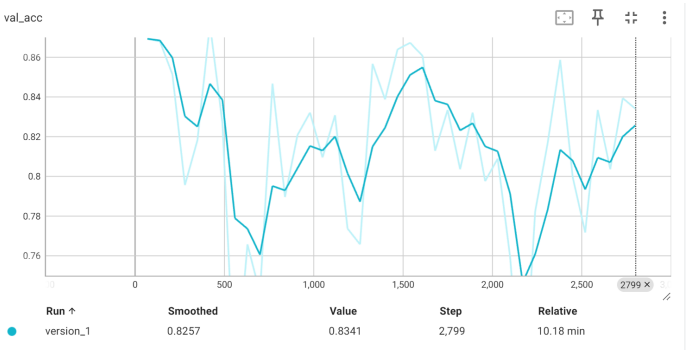

图 11: 训练过程 val acc 的曲线

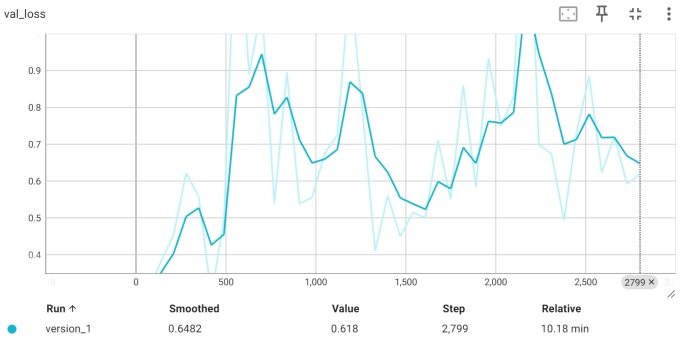

图 12: 训练过程 val loss 的曲线

2.3.4 test.py

可同时处理多个文件夹和多个模型的的评估，且可通过 stack 批处理图像，还可以通过 convert 函数预测单通道的图像。其核心代码见 Listing 5。

```
cm = confusion_matrix(y_true, y_pred) # 计算混淆矩阵

# 计算准确率、召回率、精确率和 F1 分数
tn, fp, fn, tp = cm.ravel()
accuracy = (tp + tn) / (tp + tn + fp + fn)
precision = tp / (tp + fp)
recall = tp / (tp + fn)
f1_score = 2 * (precision * recall) / (precision + recall)
```

Listing 5: test.py 核心代码

预测数据可视化：

对于二分类问题，混淆矩阵通常呈现如下形式：

	Predicted	
Actual	Negative	Positive
Negative	TN	FP
Positive	FN	TP

- **TN（True Negative）**：真负类，实际为负类（非暴力图像），预测也为负类。
- **FP（False Positive）**：假正类，实际为负类（非暴力图像），但预测为正类（暴力图像）。
- **FN（False Negative）**：假负类，实际为正类（暴力图像），但预测为负类（非暴力图像）。
- **TP（True Positive）**：真正类，实际为正类（暴力图像），预测也为正类。

准确率（Accuracy）：评估模型整体的正确性，反映了模型在所有样本中的表现。表示模型预测正确的比例：

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

精确率（Precision）：评估模型对正类（暴力图像）的预测准确性，避免了将太多负类（非暴力图像）错误分类为正类。

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

召回率（Recall）：评估模型对正类（暴力图像）的检出能力，避免漏掉实际为正类的样本。

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

F1 分数：综合考虑了精确率和召回率的平衡性，是一个对不平衡数据集更友好的评估指标。

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

表 2: 同源数据集测试结果

folder	Confusion Matrix	Accuracy	Precision	Recall	F1 score
同源数据集	$\begin{bmatrix} 326 & 74 \\ 17 & 383 \end{bmatrix}$	0.88625	0.83807	0.9575	0.89382
AIGC	$\begin{bmatrix} 133 & 26 \\ 6 & 125 \end{bmatrix}$	0.88966	0.82781	0.9542	0.88652
对抗样本	$\begin{bmatrix} 342 & 31 \\ 41 & 393 \end{bmatrix}$	0.91078	0.92689	0.9055	0.91608
噪声图片	$\begin{bmatrix} 437 & 36 \\ 17 & 517 \end{bmatrix}$	0.94737	0.93490	0.9682	0.95124

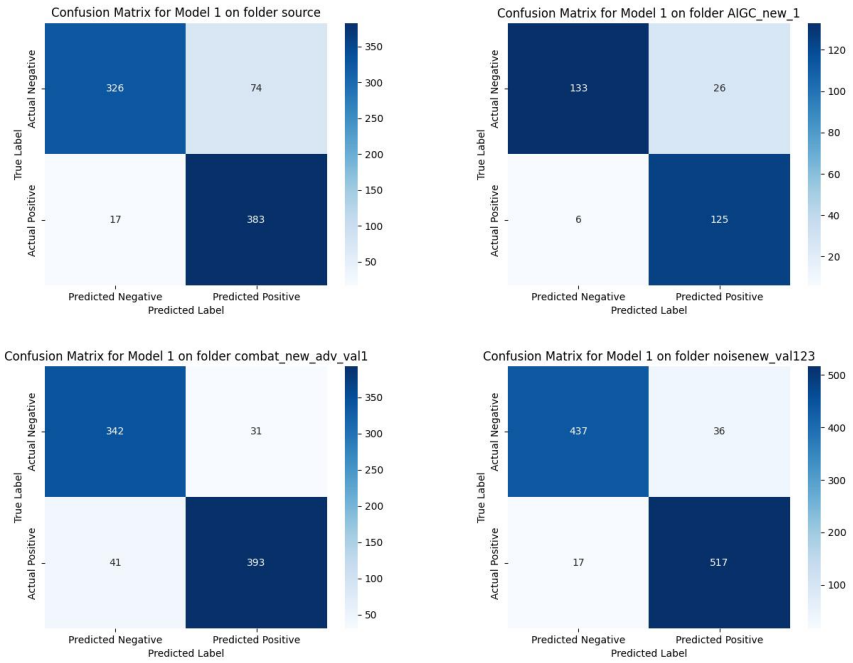

图 13: 同源数据集测试结果图

2.4 测试结果分析

同源数据集 在同源数据集上测试，结果如表 2 和图 13 所示。

2.5 接口类文件 classify.py

实现接口类 ViolenceClass，图像预处理，各类参数设置以及模型加载等初始化工作。接口类中包含初始化方法、预处理、分类方法和归一化方法，代码详见 Listing 6。

```
class ViolenceClass:
    def __init__(self, model_checkpoint_path: str, device: str
        = 'cuda:0'):
        # 设置设备
        self.device = torch.device(device if torch.cuda.
            is_available() else 'cpu')

        # 加载模型
        self.model = Resnet34ViolenceClassifier.
            load_from_checkpoint(model_checkpoint_path)
        self.model.to(self.device)
        self.model.eval()

        # 定义预处理
        self.transform = transforms.Compose([
            transforms.Resize((224, 224)),
            transforms.ToTensor(),
            #transforms.Normalize([0.485, 0.456, 0.406],
                [0.229, 0.224, 0.225])
        ])

    def classify(self, imgs : torch.Tensor) -> list:
        # 确保输入是一个torch.Tensor，并且已经在0-1范围内
        if not isinstance(imgs, torch.Tensor):
            raise TypeError("Input should be a torch.Tensor")

        # 如果输入是单张图像，扩展为批次维度
        if imgs.dim() == 3:
            imgs = torch.unsqueeze(imgs, 0)

```

```
27          # 归一化处理
28          imgs_normalized = self.normalize_images(imgs)
29
30          # 将输入tensor移动到目标设备
31          imgs_normalized = imgs_normalized.to(self.device)
32
33          preds = []
34          with torch.no_grad():
35              output = self.model(imgs_normalized)
36              predict = torch.softmax(output, dim=1)
37              _, classes = torch.max(predict, dim=1)
38              preds = classes.cpu().numpy().tolist()
39
40          # 如果输入是单张图像，则返回预测类别索引，否则返回预测
                类别索引列表
41          return preds[0] if len(preds) == 1 else preds
42
43      def normalize_images(self, imgs: torch.Tensor) -> torch.
            Tensor:
44          # 确保输入是在0-1范围内的torch.Tensor，并且是RGB格式
45          assert imgs.dim() == 4  # 假设输入是(batch_size,
                channels, height, width)
46          assert imgs.shape[1] == 3  # 假设输入是RGB图像
47
48          normalize = transforms.Normalize(mean=[0.485, 0.456,
                0.406], std=[0.229, 0.224, 0.225])
49          normalized_imgs = normalize(imgs)
```

Listing 6: 接口类 ViolenceClass

初始化方法 (___init___): 设置了设备选择，加载了预训练模型，并将其移到设备上。

预处理 (transform): 定义了图像预处理管道，包括调整大小和转换为张量操作。

分类方法 (classify): 接受输入张量并执行归一化、设备移动以及推理操作，返回预测类别索引或索引列表。

归一化方法 (normalize_images): 确保输入张量是 RGB 格式并进行归一化处理，使用指定的均值和标准差。

3 工作总结

3.1 收获、心得

深度学习模型的理解和实践

我们实践了用于暴力与非暴力图像二分类的深度学习模型，通过设计、构建、训练和调试模型，掌握了模型的基本结构和工作原理，学会了如何合理地调整模型的超参数、选择合适的激活函数和损失函数，从而提高模型的整体性能。

数据处理和增强技术

为了提高模型的泛化能力，我们应用了多种数据增强技术，不仅使用自然图像作为数据集的组成部分，还使用了 AIGC 图像、根据原自然图像生成的对抗样本和加噪图像，来增强数据集的多样性。在该过程中，我们深刻理解了数据多样性对模型训练的重要性。

模型性能的评估与改进

在模型的评估过程中，我们使用了多种指标，如准确率、精确率、召回率、F1 分数等，并通过混淆矩阵详细分析了模型的性能，这使我们能够更全面地了解模型在不同类别上的表现，从而针对性地进行改进。

团队协作的重要性

小组成员之间的合理分工和有效沟通是项目顺利进行的保障。我们通过定期的线上和线下会议，及时交流工作进展和遇到的问题，明确下一步的任务和目标，确保每个人都做好自己的任务和工作。

3.2 遇到问题及解决思路

噪声幅度选取

噪声过小则无法验证模型的鲁棒性，过大影响到模型分类的准确性评估。选取不同的噪声类型（高斯噪声、椒盐噪声），并基于实验逐步调整噪声的强度。

AIGC 风格同质化

在前期，AIGC 图像风格的同质性较高，对数据集多样性的贡献较小，作为训练集时不能使模型具有好的泛化能力，作为测试集也无法检测模型的真实性能。因此后期使用了多种 SD 模型，并通过调整生成过程中的超参数来获得更多样化的图像。

4 小组分工

张羽仪：AIGC 和对抗样本生成，操作文档撰写
花艺：噪声样本生成，操作文档撰写
徐菲悦：模型训练，操作文档撰写
彭娜：模型训练，操作文档撰写

5 课程建议

- 学期中可以适量布置一些与课程强相关的小实验，提高同学们对课程知识的理解和运用水平。
- 课堂上可以多设置一些随堂提问或小测验，驱动同学们去主动思考。