Violence Classify Project Report

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。此外，我们还要求该模型有合理的运行时间且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。

2 具体内容

2.1 实施方案

2.1.1 环境配置

由于本地没有显卡，故连接实验室服务器。创建一个新的conda虚拟环境，使用conda安装python3.8，pytorch1.8.2，torchvision==0.9.2。

```
conda create -n myenv python=3.8
conda install pytorch==1.8.2 torchvision==0.9.2 -c pytorch
```

2.1.2 训练数据集创建

通过连接https://jbox.sjtu.edu.cn/l/x1rzkN 获取训练及评测数据共8857张图像，在代码同级目录下创建data文件夹，数据集分别放在data/train和data/test。

2.1.3 训练代码实现

结合支持文档所提供的参考代码，实现dataset.py，model.py，train.py三个文件，放在和data同级目录下。

2.1.4 接口类文件实现

实现接口类文件classify.py，其定义了接口类ViolenceClass，ViolenceClass提供一个接口函数classify。在接口函数中实现了实现模型加载、模型推理等逻辑。

2.1.5 测试脚本编写

该脚本从data/test中读取jpg文件，由PIL库读入并归一化到0-1的tensor。将一个`n*3*224*224`的pytorch tensor作为输入，得到长度为`n`的python列表（每个值为对应的预测类别，即整数0或1）。

2.2 核心代码分析

2.2.1 classify.py

```
import torch
import numpy as np
from torch import nn
from torchvision import models
from torchvision import transforms
from torchmetrics import Accuracy
from pytorch_lightning import LightningModule

class ViolenceClass:
    def __init__(self):
        # 加载预训练模型
        ckpt_path = "resnet18_pretrain_test-epoch=02-val_loss=0.03.ckpt"
        self.model =
ViolenceClassifier.load_from_checkpoint(ckpt_path).to('cuda:0')

    def classify(self, tensor_imgs):
        #设置处于评估模式
        self.model.eval()

        with torch.no_grad():
            tensor_imgs = tensor_imgs.to('cuda:0')
            outputs = self.model(tensor_imgs)
            _, predicted = torch.max(outputs, 1)
            print(predicted.tolist())

        return predicted.tolist()

#ViolenceClass类中需要使用ViolenceClassifier的load_from_checkpoint方法故此处定义
ViolenceClassifier类

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.learning_rate = learning_rate
```

```
37          self.loss_fn = nn.CrossEntropyLoss()  # 交叉熵损失
38          self.accuracy = Accuracy(task="multiclass", num_classes=2)
39          #后续相关方法此处省略
```

`ViolenceClass`类用于加载预训练模型并进行分类。它包含：

- `__init__`方法：初始化类实例，加载预训练的模型检查点，并将其移动到GPU上。
- `classify`方法：进行模型推理，将输入图像张量（`tensor_imgs`）传入模型，获取预测结果，并打印和返回预测类别列表。

2.2.2 dataset.py

`CustomDataset`类继承自`torch.utils.data.Dataset`，用于封装数据集的加载逻辑。它包含：

- `__init__`方法：初始化数据集，接受一个`split`参数来区分训练集、验证集或测试集，并根据`split`应用不同的图像变换。
- `__len__`方法：返回数据集中样本的数量。
- `__getitem__`方法：根据索引`index`加载并返回单个数据点（图像和标签）。图像通过PIL库加载，并应用预定义的变换。标签从图像路径中解析出来。

`CustomDataModule`类继承自`pytorch_lightning.LightningDataModule`，用于封装数据模块的逻辑。它包含：

- `__init__`方法：初始化数据模块，定义批量大小`batch_size`和工作进程数`num_workers`。
- `setup`方法：在这个阶段，数据集被分割并准备就绪。根据`stage`（训练、验证或测试）创建相应的数据集实例。
- `train_dataloader`、`val_dataloader`和`test_dataloader`方法：这些方法返回相应的`DataLoader`实例，用于在训练、验证和测试阶段加载数据。`DataLoader`负责批处理、打乱（仅训练集）和多进程加载。

2.2.3 model.py

- 构造函数中首先调用父类 `LightningModule` 的构造函数。
- 加载预训练的 ResNet-18 模型，并将其最后的全连接层替换为新的全连接层，以适应 `num_classes` 类别的分类任务。
- 定义了模型的学习率、损失函数（交叉熵损失）和准确率度量。
- `forward` 方法定义了模型的前向传播过程，即输入 `x` 通过模型的处理。
- 定义了模型训练时使用的优化器，这里是使用 Adam 优化器，学习率由构造函数传入。
- `training_step` 方法定义了单次训练的逻辑，包括前向传播、计算损失，并记录训练损失。
- `validation_step` 方法定义了单次验证的逻辑，与训练步骤类似，但增加了准确率的计算和记录。对于二分类问题，使用 sigmoid 函数将 logits 转换为概率，并阈值化处理为 0 或 1。
- `test_step` 方法定义了单次测试的逻辑，与验证步骤类似，但只记录准确率。

2.2.4 train.py

- 创建了一个 `CustomDataModule` 实例，它将负责加载和准备数据。
- 定义了一个 `ModelCheckpoint` 回调，用于在训练过程中保存模型的状态。它将根据验证损失 (`val_loss`) 来监控并保存最佳的模型（`save_top_k=1` 表示只保存最佳的一个模型），文件名包含 epoch 数和验证损失，模式设置为最小化损失 (`mode='min'`)。
- 创建了一个 `TensorBoardLogger` 实例，用于记录训练和验证过程中的指标，这些指标可以被 TensorBoard 可视化。
- 创建了一个 `Trainer` 实例
- 创建了 `ViolenceClassifier` 模型的实例，传入学习率作为参数，调用 `Trainer` 的 `fit` 方法开始训练过程，传入模型和数据模块。

2.3 测试实验

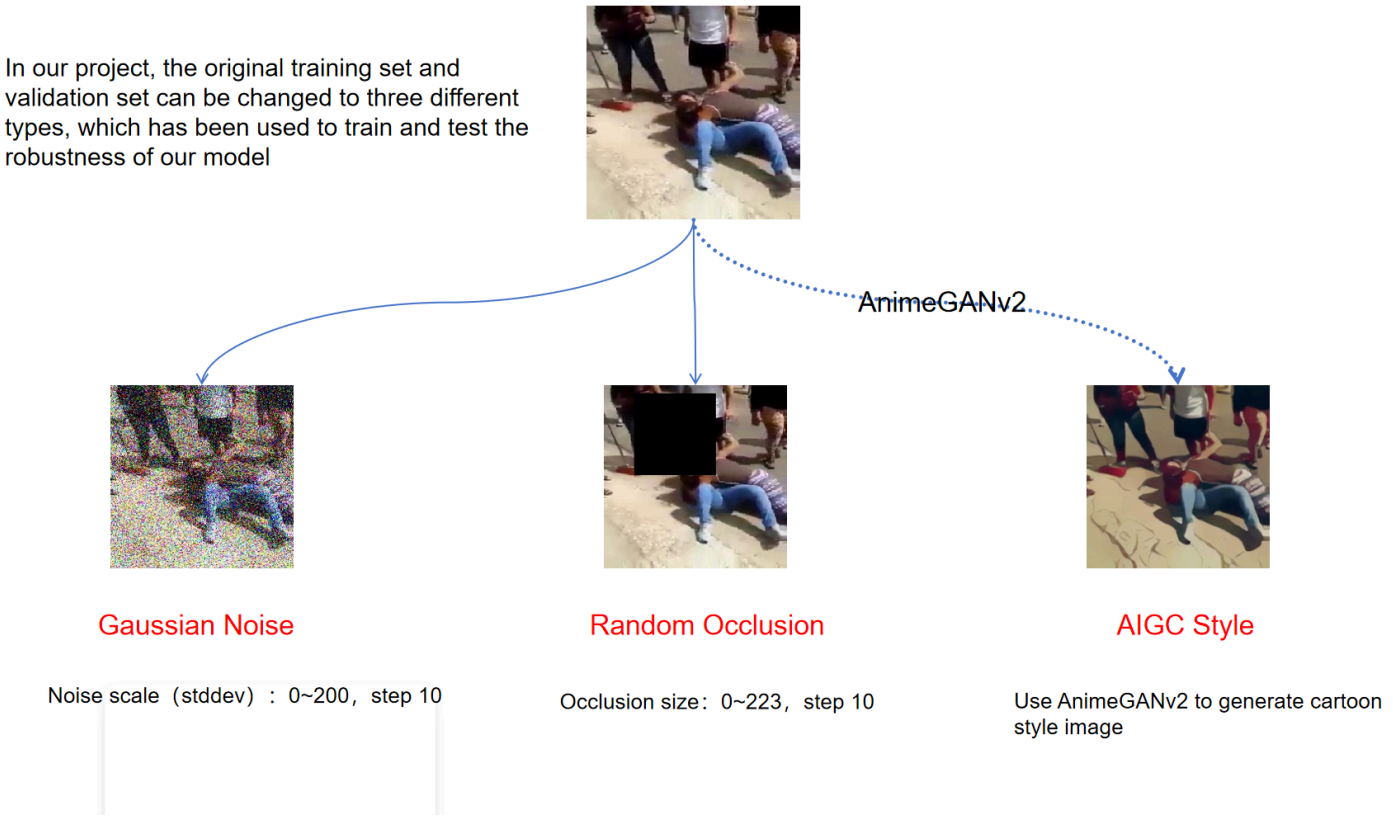

fig 2.3 Three different ways to handle robustness in our project

2.3.1 验证数据集

为了测试模型的效果，我们会分别将训练好的模型应用应用在不同种类的验证集：val、gaussian_noise_val、occlusion_val、aigc_val、real aigc

1. val

   这是已经给出的验证数据

2. guassian_noise_val

   这是在val的基础上，叠加一定强度的高斯噪声（噪声强度可调）的数据

3. occlusion_val

   在val的基础上，随机遮挡一部分图片的数据（遮挡的大小可调）

4. aigc_style_val

   在val的基础上，我们采用AnimeGANv2对图片进行风格转化，一定程度上还原了AIGC生成的图片效果。

5. real_aigc

我们手工在Stable diffusion网站上通过仔细打磨prompt得到的数据，可以切实地反映模型在aigc方面的泛化能力

其中，高斯噪声和随机遮挡具有可调节性，所以我们做了一定的封装，统一用`add_noise_to_image`实现，以下是核心代码

```python
def add_noise_to_image(image, noise_type="occlusion", occlusion_size=(100, 100), fixed_gaussian_noise=0, fixed_gaussian_sttdev=25):
    """
    Add noise to a single image.

    :param image: Input image as a NumPy array.
    :param noise_type: Type of noise to add ("occlusion" or "gaussian_noise").
    :param occlusion_size: Size of the occlusion patch if noise_type is "occlusion".
    :return: Image with added noise.
    """
    noisy_image = image.copy()

    if noise_type == "occlusion":
        # Add occlusion (black patch) to the image
        x_start = np.random.randint(0, image.shape[1] - occlusion_size[1])
        y_start = np.random.randint(0, image.shape[0] - occlusion_size[0])
        noisy_image[y_start:y_start + occlusion_size[0], x_start:x_start + occlusion_size[1], :] = 0
    elif noise_type == "gaussian_noise":
        # Add Gaussian noise to the image
        noise = np.random.normal(fixed_gaussian_noise, fixed_gaussian_sttdev, image.shape).astype(np.float32)
        noisy_image = cv2.add(noisy_image.astype(np.float32), noise)
        noisy_image = np.clip(noisy_image, 0, 255).astype(np.uint8)
    else:
        raise ValueError("Unsupported noise type. Use 'occlusion' or 'gaussian'.")
    return noisy_image
```

从而可以通过循环传入一定步长的噪声参数，来进行更加全面综合的分析测试。

图片的风格转换是基于开源项目AnimeGANv2对val进行修改的，大致的效果如下：

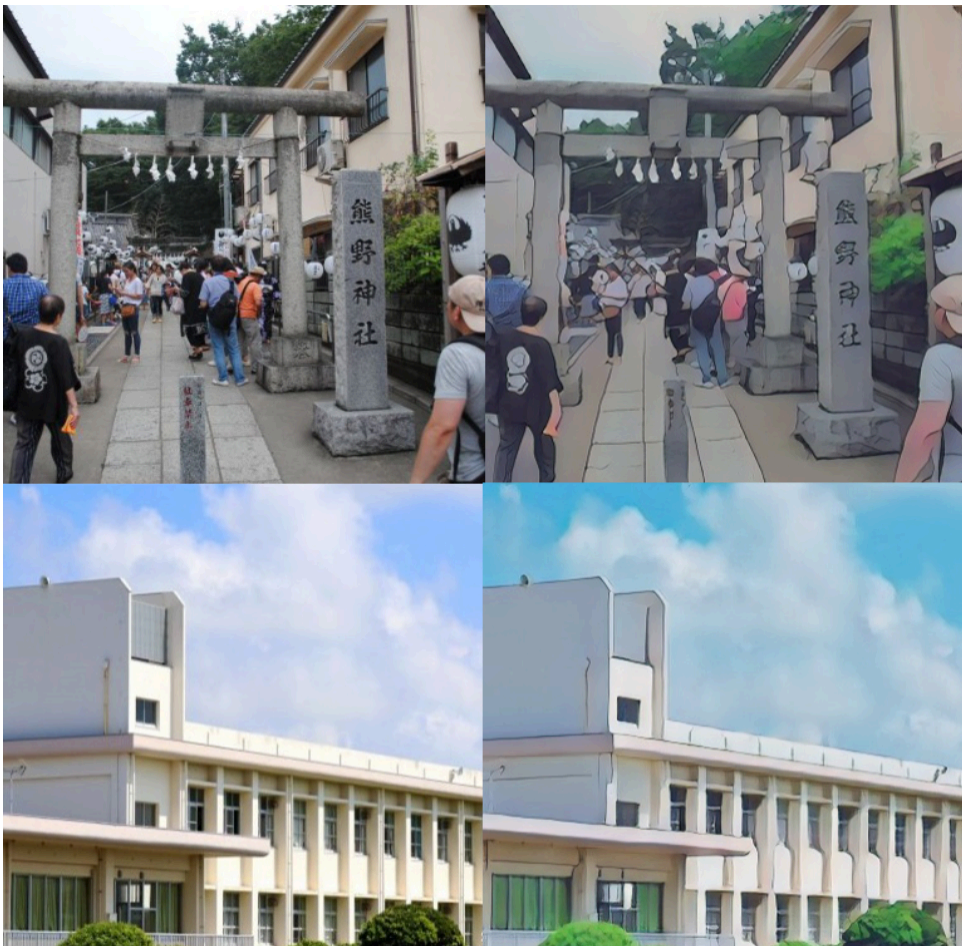

fig 2.3.1 AnimeGANv2 Renderings

本实验主要用AnimeGANv2对训练集和测试集val进行风格转化。

2.3.2 测试流程以及改进

在整个过程中，我们一共训练并测试了三个模型，分别为只学习过train_dataset、学习过train_dataset + train_dataset with gaussian noise、学习过train_dataset + train_dataset with noise + train_dataset with aigc style（以下分别简称称orignal、gaussian、aigc&gaussian）

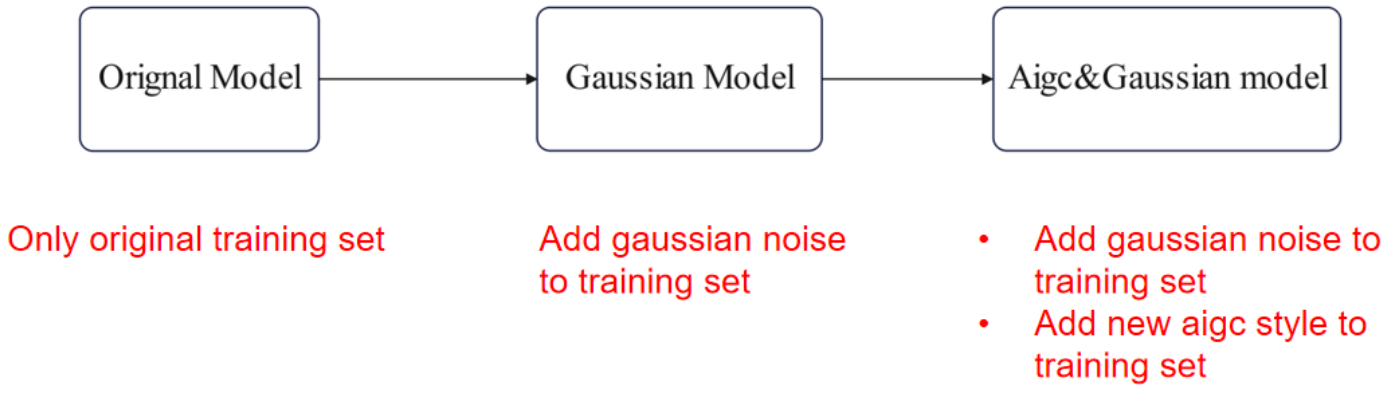

fig 2.3.2 The process of model optimization

2.3.2.1 orignal model

只给模型学习训练集，然后分别在三个数据集上进行测试

1. val

   测试的结果为0.960252935862692，是一个相当高的结果

2. val with occlusion noise

   我们对val数据集进行随机遮挡，并设置好遮挡大小的改变步长（从0*0到223*223，步长为10），测试结果如下

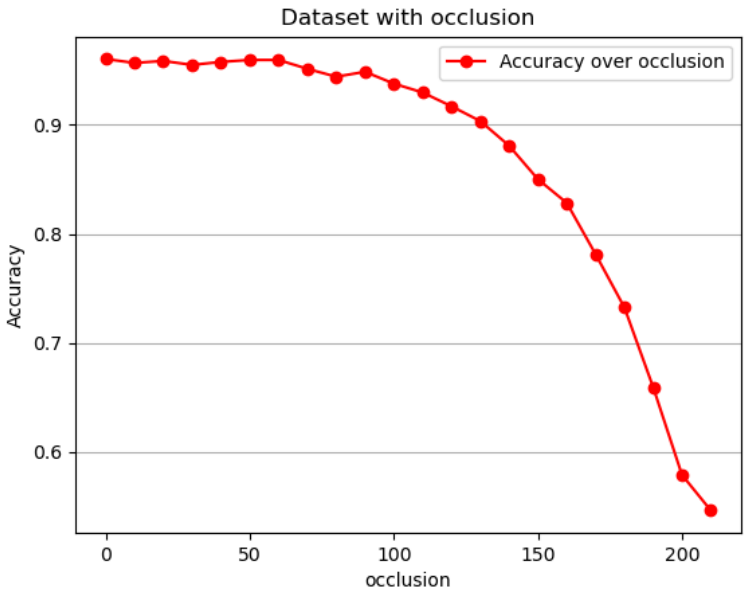

fig 1 Accuracy over occlusion size (Origianl model)

3. val with gaussian noise

   接着我们尝试对添加的高斯噪声的数据进行测试，均值设为默认50，方差设置为0~200，步长为10，得到的结果如下

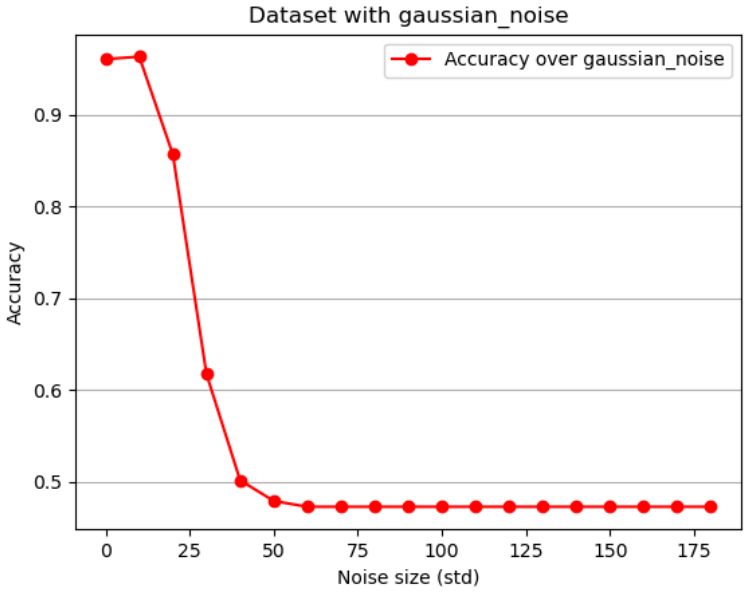

fig 2 Accuracy over gaussian noise (Original model)

可见，模型对高斯噪声具有明显的敏感性，需要改进训练方案。

4. val with aigc style

   通过AnimeGANv2对val数据进行风格转化，再进行测试，得到结果0.7633242999096658

5. real aigc

   测试结果为70.0932428%

2.3.2.2 gaussian model

在上一个模型的基础上，我们准备给训练集添加噪声（std=100），并混合在原本的数据中。

1. val

   测试结果为0.986449864498645，比original model还高。我们认为，可能原因是加入高斯噪声的训练集和原本的训练集中的数据有很大部分的特征重叠。对模型来说，相当于对突触的特征学习了多次，在一定程度上学习得更加“充分”了。所以效果会好一点。

2. val with occlusion noise

   在添加了随机遮挡的val上进行测试，与original同样的步长设置，得出的结果如下：

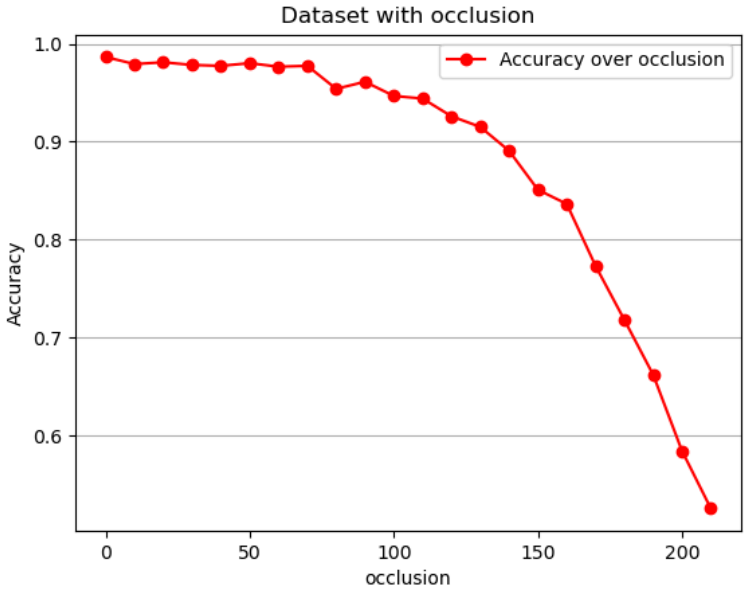

fig 3 Accuracy over occlusion size (Gaussian model)

   可见，经过高斯噪声训练的模型，对occlusion的鲁棒性能没有太大的影响。

3. val with gaussian noise

   在添加了高斯噪声的val进行测试，步长设置和original model一致，结果如下：

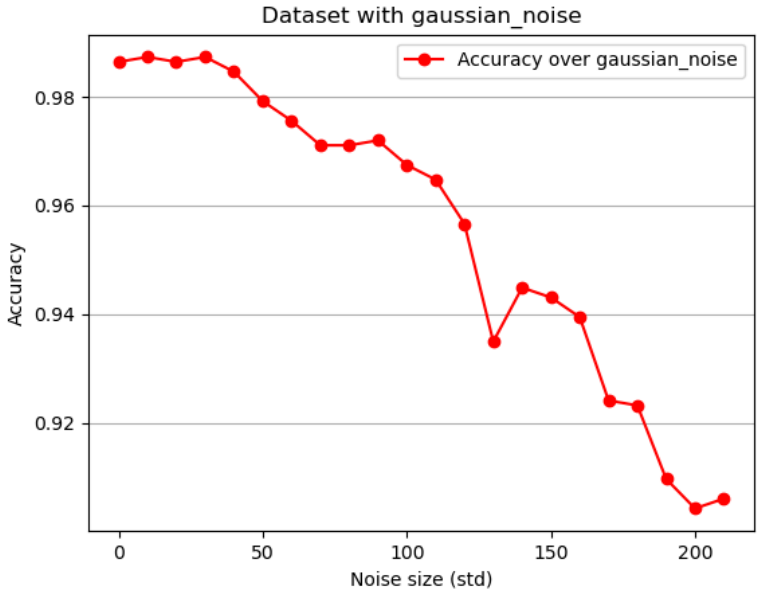

fig 4 Accuracy over gaussian noise (Gaussian model)

可以看出，模型对高斯噪声的稳定性增强了，基本稳定在90%以上。

4. val with aigc style

   改变val数据的风格，得到具有类似aigc风格的图片，进行测试。

   测试结果为0.9284462511291779

   测试结果相较于original有明显提高，分析原因，可能是AIGC图片具有高斯噪声相似的特征，在一些地方同样会模糊处理。

5. real aigc

   测试结果为0.7889221556886228，说明在真实的aigc场景下，模型仍然有改进的空间。

2.3.2.3 aigc&gaussian model

我们采用AnimeGANv2对训练集进行优化，调整了风格，并加入训练。

1. val

   测试结果为0.9846431797651309

2. val with occlusion noise

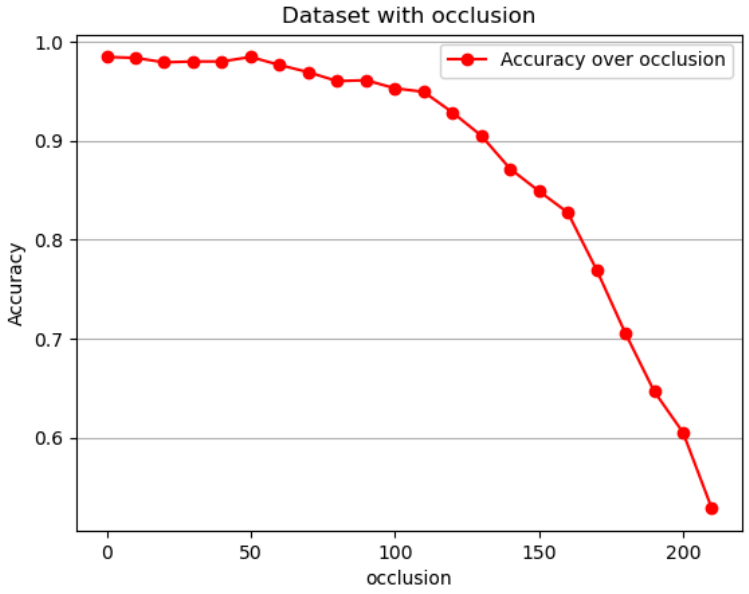

fig 5 Accuracy over occlusion size (aigc&gaussian model)

3. val with gaussian noise

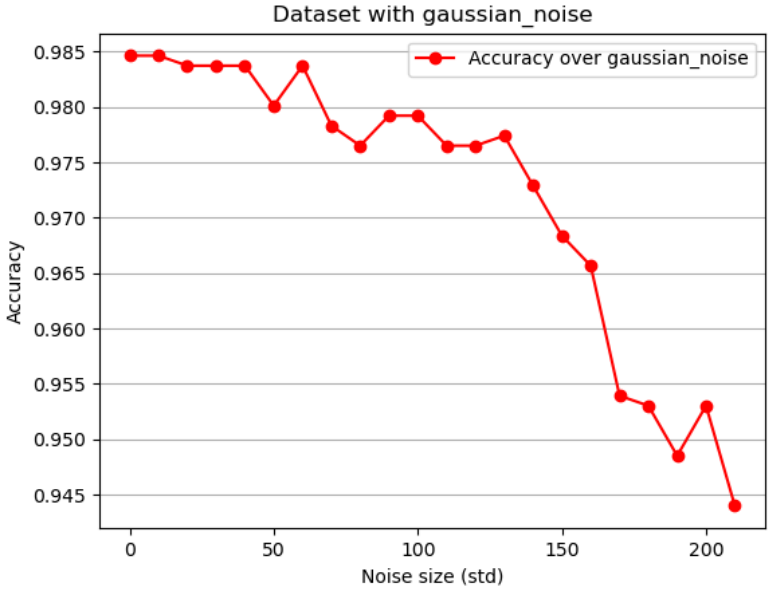

fig 6 Accuracy over gaussian noise (aigc&gaussian model)

4. val with aigc style

测试结果为0.9774164408310749

5. real aigc

测试结果为0.8208982035928144，相较于gaussian model的结果有一些提高，但是仍然存在缺陷，分析原因，可能是实际情况下的aigc图片的特征变化是多样的，仅仅凭借修改模型风格难以完全学习到aigc violence的规律。

3 工作总结

3.1 收获 & 心得

（1）通过这次实验，我熟悉了构建一个模型整体的流程：从获取数据集到训练模型到验证，设置checkpoint来保存模型。在测试过程中使用load_from_checkpoint方法来导入模型进行测试。

3.2 遇到问题及解决思路

（1）第一次train的时候无法运行，查看报错得知pytorch_lightning版本问题，当前版本为2.2.2，uninstall后重新安装1.6.0版本

（2）编写脚本测试得出结果全都是1，检查输入发现了问题。没有将输入的值归一化至0到1，导致输入的数很多大于1

（3）显示数据同时位于cuda:0和cuda:1两个卡，但是找遍所有的文件发现都是把tensor_img放到了0号卡上，没有找到指定一号卡的变量。最后利用排除法发现只有可能是model默认在1号卡上面跑，因此把model移到0号卡上面。

（4）参考代码中给的脚本是直接使用CustomDataModule类来从/data/test中导入图片的，但要求classify却是以tensor作为输入，虽然图片转换成向量不是一个很难的问题，但是这样就不能使用pytorch_lightning的Trainer类的test方法了（此方法是一个端到端的方法：即以图片作为输入，准确率和loss作为输出）。这个问题困扰了我很久，最后发现只要将tensor作为model的输入即可（之后分析造成困扰的原因还是因为对LightningModule源码不熟悉，不清楚需要接受哪些作为输入）

（5）中途报错forward方法封装的conv2d接受的参数类型不匹配，原因是没有把输入的python的list转化为pytorch的tensor导致。

4 课程建议

希望老师能够多多指导学生完成作业。其实不一定要再课堂上学到东西，在完成项目的过程中也能学到很多，而且效率更高。所以，我期待之后的课程能够更加密切地与项目结合。