实验二

202218018670003 马天行

00 概述

实验内容是使用Resnet18网络实现猫狗图片分类任务，文件结构等信息可以在前置文档《实验项目结构概述》中看到。

完整代码和文档见仓库：https://github.com/Noitolar/CourseDL.git

01 数据集和预处理

使用Kaggle官方的猫狗分类数据集，在实验之前已经将带有标签的训练集的95%划分到本实验的训练集，另外5%作为验证集。在攻击函数中没有默认的预处理配置项，具体预处理配置详见第06节。

```python
import torchvision.datasets as vdatasets
import torchvision.transforms as trans
import utils.cv as ucv


def get_dataset(config: ucv.config.ConfigObject):
    if config.dataset == "mnist":
        ...
    else:
        assert config.trn_preprocess is not None
        assert config.val_preprocess is not None
        trn_set = vdatasets.ImageFolder(root=f"./datasets/{config.dataset}/train",
transform=config.trn_preprocess)
        val_set = vdatasets.ImageFolder(root=f"./datasets/{config.dataset}/validation",
transform=config.val_preprocess)
        return trn_set, val_set
```

02 神经网络模型

基于torchvision官方提供的resnet18网络，可以自定义配置输出的通道数量（例如在进行Mnist分类任务时应该将输入通道数设为1），以及预训练权重，其他不做改动。

```python
import torch
import torch.nn as nn
import torchvision.models as vmodels
```

```
class Resnet18Classifier(nn.Module):
    def __init__(self, num_classes, num_channels=3, from_pretrained=None):
        super().__init__()
        if from_pretrained == "default":
            self.resnet = vmodels.resnet18(weights=vmodels.ResNet18_Weights.DEFAULT)
        elif from_pretrained == "imagenet":
            self.resnet = vmodels.resnet18(weights=vmodels.ResNet18_Weights.IMAGENET1K_V1)
        else:
            self.resnet = vmodels.resnet18(weights=None)
        self.resnet.conv1 = nn.Conv2d(num_channels, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3),
bias=False)
        self.resnet.fc = nn.Linear(512, num_classes)

    def forward(self, inputs: torch.Tensor):
        outputs = self.resnet(inputs)
        return outputs
```

03 模型操作

第03节到第05节实验一完全一致，如果不想看可以直接跳转到第06节。

这个类用于对模型进行一些高级操作，成员包括：

- 配置项
- 记录器（用于记录训练过程中的指标以及进行日志管理）
- 设备
- 神经网络模型
- 损失函数

类方法包括：

- 日志记录：调用成员记录器的方法，将指定信息录入日志，并同步显示在终端
- 配置记录：按照固定格式将实验配置记录到日志，并同步显示在终端
- 数据转移：将内存中的张量数据转移到配置的设备的内存/显存中
- 正向调用：根据输入和标签输出预测值和损失值，如果没有输入标签则只输出预测值

在初始化成员对象之前会先设置全局随机数种子，保证实验结果可以复现。

```
import torch
import torch.nn as nn
import torchvision.transforms as trans
import time
import transformers as tfm
import utils.cv as ucv
```

```python
class ModelHandlerCv(nn.Module):
    def __init__(self, config: ucv.config.ConfigObject):
        super().__init__()
        self.config = config
        tfm.set_seed(config.seed)
        self.recorder = ucv.recorder.Recorder(config.log_path)
        self.device = config.device
        self.model = config.model_class(**config.model_params).to(config.device)
        self.criterion = config.criterion_class(**config.criterion_params)

    def log(self, message):
        self.recorder.audit(message)

    def log_config(self):
        self.log(f"\n\n[+] exp starts from: {time.strftime('%Y-%m-%d %H:%M:%S', time.localtime())}")
        for config_key, config_value in self.config.params_dict.items():
            if config_value is None:
                continue
            elif config_key.endswith("_class"):
                self.log(f"[+] {config_key.replace('_class', '')}: {config_value.__name__}")
            elif config_key.endswith("_params") and isinstance(config_value, dict):
                for param_key, param_value in config_value.items():
                    self.log(f"    [-] {config_key.replace('_params', '')}.{param_key}: {param_value}")
            elif isinstance(config_value, trans.transforms.Compose):
                self.log(f"[+] {config_key}:")
                for index, value in enumerate(str(config_value).replace(" ", "").split("\n")[1:-1]):
                    self.log(f"    [-] {index:02d}: {value}")
            else:
                self.log(f"[+] {config_key}: {config_value}")

    def device_transfer(self, data):
        if isinstance(data, torch.Tensor):
            data = data.to(self.device)
        if isinstance(data, dict):
            data = {key: value.to(self.device) for key, value in data.items()}
        return data

    def forward(self, inputs: torch.Tensor, targets=None):
        inputs = self.device_transfer(inputs)
        targets = self.device_transfer(targets)
        preds = self.model(inputs)
        loss = self.criterion(preds, targets) if targets is not None else None
        return preds, loss
```

04 记录器

这个类用于记录模型训练过程中的一些指标，以及日志管理的相关功能，成员对象包括：

- 累计准确率
- 累计损失值
- 累计样本数
- 日志记录器

成员方法包括：

- 计算并更新累计准确率、损失值、样本数
- 还原成员变量
- 返回平均准确率和损失值
- 日志录入

```python
import numpy as np
import sklearn.metrics as metrics
import logging
import os


class Recorder:
    def __init__(self, logpath):
        self.accumulative_accuracy = 0.0
        self.accumulative_loss = 0.0
        self.accumulative_num_samples = 0
        self.logger = logging.getLogger(__name__)
        self.logger.setLevel(logging.DEBUG)
        self.logger.addHandler(logging.StreamHandler(stream=None))
        if logpath is not None:
            if not os.path.exists(os.path.dirname(logpath)):
                os.makedirs(os.path.dirname(logpath))
                logfile = open(logpath, "a", encoding="utf-8")
                logfile.close()
            self.logger.addHandler(logging.FileHandler(filename=logpath, mode="a"))

    def update(self, preds, targets, loss):
        assert len(preds) == len(targets)
        num_samples = len(preds)
        preds = np.array([pred.argmax() for pred in preds.detach().cpu().numpy()])
        targets = targets.detach().cpu().numpy()
        self.accumulative_accuracy += metrics.accuracy_score(y_pred=preds, y_true=targets) * num_samples
        self.accumulative_loss += loss * num_samples
        self.accumulative_num_samples += num_samples

    def clear(self):
        self.accumulative_accuracy = 0.0
        self.accumulative_loss = 0.0
        self.accumulative_num_samples = 0

    def accuracy(self):
        accuracy = self.accumulative_accuracy / self.accumulative_num_samples
        loss = self.accumulative_loss / self.accumulative_num_samples
        return accuracy, loss

    def audit(self, msg):
        self.logger.debug(msg)
```

05 模型训练

用于训练和评估模型的类，成员对象包括：

- 模型操作器
- 优化器
- 学习率调整策略（可选）

成员方法包括：

- 训练：训练模型一轮，会调用模型操作器（ModelHandler）的记录器（Recorder）计算训练时的准确率和损失，并在本轮结束时返回训练报告并重置记录器
- 验证：验证模型，没有反向传播过程，并且不计算梯度以节省显存和算力

```
import torch
import tqdm



class Trainer:
    def __init__(self, handler):
        self.handler = handler
        self.optimizer = handler.config.optimizer_class(handler.model.parameters(),
**handler.config.optimizer_params)
        self.scheduler = handler.config.scheduler_class(self.optimizer, **handler.config.scheduler_params) if
handler.config.scheduler_class is not None else None

    def train(self, loader):
        self.handler.train()
        for inputs, targets in tqdm.tqdm(loader, desc=f"    [-] training", delay=0.2, leave=False, ascii="-
>"):
```

```
            preds, loss = self.handler(inputs, targets)
            self.handler.recorder.update(preds, targets, loss)
            self.optimizer.zero_grad()
            loss.backward()
            self.optimizer.step()
        accuracy, loss = self.handler.recorder.accuracy()
        self.handler.recorder.clear()
        if self.scheduler is not None:
            self.scheduler.step()
        report = {"loss": loss, "accuracy": accuracy}
        return report

    @torch.no_grad()
    def validate(self, loader):
        self.handler.eval()
        for inputs, targets in tqdm.tqdm(loader, desc=f"    [-] validating", delay=0.2, leave=False, ascii="->"):
            preds, loss = self.handler(inputs, targets)
            self.handler.recorder.update(preds, targets, loss)
        accuracy, loss = self.handler.recorder.accuracy()
        self.handler.recorder.clear()
        report = {"loss": loss, "accuracy": accuracy}
        return report
```

06 实验主函数

配置、对象初始化、训练和评估。

数据集的基本与处理是缩放到固定尺寸、张量化以及标准化，训练集在此基础上还有一定的数据增强处理，包括图片裁剪以及随机水平翻转。模型使用自imagenet上训练的预训练权重。批大小为32，共训练8轮，SGD优化器初始学习率为0.001，每4轮下降至原来的十分之一。

```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.utils.data as tdata
import torchvision.transforms as trans
import utils.cv as ucv

if __name__ == "__main__":
    config = ucv.config.ConfigObject()

    config.model_class = ucv.nnmodels.Resnet18Classifier
    config.model_params = {"num_classes": 2, "num_channels": 3, "from_pretrained": "imagenet"}
    config.device = "cuda:0"
    config.criterion_class = nn.CrossEntropyLoss
    config.criterion_params = {}
    config.log_path = "./logs/dogs_vs_cats.resnet18.log"

    config.dataset = "dogs_vs_cats"
    config.seed = 0
    config.trn_preprocess = trans.Compose([
        trans.Resize((256, 256)),
        trans.RandomCrop((224, 224)),
        trans.RandomHorizontalFlip(),
        trans.ToTensor(),
        trans.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.2225))
    ])
    config.val_preprocess = trans.Compose([
        trans.Resize((256, 256)),
        trans.ToTensor(),
        trans.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.2225))
    ])
    config.batch_size = 32
    config.num_epochs = 8
    config.optimizer_class = optim.SGD
    config.optimizer_params = {"lr": 0.001, "momentum": 0.9, "nesterov": True}
    config.scheduler_class = optim.lr_scheduler.StepLR
    config.scheduler_params = {"step_size": 4, "gamma": 0.1}
    config.checkpoint_path = "./checkpoints/dogs_vs_cats.resnet18.pt"

    handler = ucv.handler.ModelHandlerCv(config)
    handler.log_config()

    trn_set, val_set = ucv.dataset.get_dataset(config)
    trn_loader = tdata.DataLoader(trn_set, batch_size=config.batch_size, shuffle=True)
    val_loader = tdata.DataLoader(val_set, batch_size=config.batch_size * 8)
    trainer = ucv.trainer.Trainer(handler)

    best_val_accuracy = 0.0
    for epoch in range(config.num_epochs):
        handler.log("    " + "=" * 40)
        trn_report = trainer.train(trn_loader)
        handler.log(f"    [{epoch + 1:03d}] trn-loss: {trn_report['loss']:.4f} --- trn-acc: {trn_report['accuracy']:.2%}")
        val_report = trainer.validate(val_loader)
        handler.log(f"    [{epoch + 1:03d}] val-loss: {val_report['loss']:.4f} --- val-acc: {val_report['accuracy']:.2%}")
        if val_report["accuracy"] > best_val_accuracy:
            best_val_accuracy = val_report["accuracy"]
            if config.checkpoint_path is not None:
                torch.save(handler.model.state_dict, config.checkpoint_path)
    handler.log(f"[=] best-val-acc: {best_val_accuracy:.2%}")
```

07 实验结果

实验日志如下，包含了本次实验的配置以及实验结果，最高验证集准确率为98.56%。

```
[+] exp starts from: 2023-04-08 23:34:21
[+] model: Resnet18Classifier
    [-] model.num_classes: 2
    [-] model.num_channels: 3
    [-] model.from_pretrained: imagenet
[+] device: cuda:0
[+] criterion: CrossEntropyLoss
[+] log_path: ./logs/dogs_vs_cats.resnet18.log
[+] dataset: dogs_vs_cats
[+] seed: 0
[+] trn_preprocess:
    [-] 00: Resize(size=(256,256),interpolation=bilinear,max_size=None,antialias=None)
    [-] 01: RandomCrop(size=(224,224),padding=None)
    [-] 02: RandomHorizontalFlip(p=0.5)
    [-] 03: ToTensor()
    [-] 04: Normalize(mean=(0.485,0.456,0.406),std=(0.229,0.224,0.2225))
[+] val_preprocess:
    [-] 00: Resize(size=(256,256),interpolation=bilinear,max_size=None,antialias=None)
    [-] 01: ToTensor()
    [-] 02: Normalize(mean=(0.485,0.456,0.406),std=(0.229,0.224,0.2225))
[+] batch_size: 32
[+] num_epochs: 8
[+] optimizer: SGD
    [-] optimizer.lr: 0.001
    [-] optimizer.momentum: 0.9
    [-] optimizer.nesterov: True
[+] scheduler: StepLR
    [-] scheduler.step_size: 4
    [-] scheduler.gamma: 0.1
[+] checkpoint_path: ./checkpoints/dogs_vs_cats.resnet18.pt
    ========================================
    [001] trn-loss: 0.2173 --- trn-acc: 90.49%
    [001] val-loss: 0.1256 --- val-acc: 94.80%
    ========================================
    [002] trn-loss: 0.0862 --- trn-acc: 96.55%
    [002] val-loss: 0.0705 --- val-acc: 97.36%
    ========================================
    [003] trn-loss: 0.0627 --- trn-acc: 97.63%
    [003] val-loss: 0.0622 --- val-acc: 97.52%
    ========================================
    [004] trn-loss: 0.0517 --- trn-acc: 97.99%
    [004] val-loss: 0.0604 --- val-acc: 97.84%
    ========================================
    [005] trn-loss: 0.0369 --- trn-acc: 98.67%
    [005] val-loss: 0.0514 --- val-acc: 98.08%
    ========================================
    [006] trn-loss: 0.0308 --- trn-acc: 98.85%
    [006] val-loss: 0.0474 --- val-acc: 98.32%
    ========================================
    [007] trn-loss: 0.0301 --- trn-acc: 98.88%
    [007] val-loss: 0.0498 --- val-acc: 98.32%
    ========================================
    [008] trn-loss: 0.0297 --- trn-acc: 98.92%
    [008] val-loss: 0.0419 --- val-acc: 98.56%
[=] best-val-acc: 98.56%
```