**西安电子科技大学**

**智能系统专业实验 课程实验报告**

**实验名称 实验6-表情识别**

人工智能 学院 2120012 班

成 绩

姓名 张靖哲 学号 21009200835

实验日期 2024 年 5 月 23日

|  |
| --- |
| **实验报告内容基本要求及参考格式**   1. 实验目的   1.1 了解表情识别数据集  1.2 掌握 ResNet18 网络的结构  1.3 完成表情识别数据集的分类任务  1.4 完成模型的剪枝操作  1.5 完成表情识别任务的部署并调用摄像头进行表情识别   1. 实验所用仪器（或实验环境）   Nvidia Jetson TX1 嵌入式系统；   1. 准备工作   在项目文件中创建 dataset.py 文件，用于构建小批量训练集与测试集；  创建 model.py 文件，用于构建训练模型；  创建 utils.py 文件，用于存放训练用的各种脚本以及函数；  创建 main.py 文件，用于进行模型的训练与测试；  创建 logs 文件夹，用于保存 tensorboard 训练日志；  创建 config.json 文件，用于保存训练用的超参数；  创建 save\_path 文件夹，用于保存训练模型参数。   1. 实验基本原理及步骤（模型描述、理论描述或方案描述）   在计算机上完成模型的训练，并保存网络的参数文件。之后在嵌入式开发板上进  行模型的部署，完成模型的测试，并调用摄像头完成实时表情识别。  基本原理  1. 构建小批量训练集和测试集  本实验所用的表情识别数据集是从表情识别公开数据集 RAF-DB 中抽取部  分 数 据 所 构 成 的 ， 共 有 7 种 表 情 类 别 ， 分 别 是 ’Anger’ 、‘Disgust’、’Fear’、’Happiness’、’Neutral’、 ‘Sadness’、‘Surprise’ 每个类别的图像  数据分布与真实的 RAF-DB 数据集类似。  本实验所用的表情识别数据集共有 3034 张训练图像和 779 张测试图像。为  了构建小批量训练集和测试集，首先需要在 dataset.py 文件中撰写得到数据图像  位置与标签的函数。  2. 模型文件撰写：  本实验所用的模型是 Resnet18，在构建模型时，需要将模型的预训练文件放  入项目内。  3. 训练过程：  使用Adam或SGD优化器和自定义的重要性评估进行稀疏训练和剪枝，并在训练过程中记录和显示损失、准确率、参数和计算量等信息。  4. 动态显示：  通过调用 train\_model 函数并在训练过程中通过TensorBoard动态显示损失、准确率、参数量和计算量等信息。  代码步骤  1、导入必要的库和模块：  包括 torch, utils, numpy, random, os, time, transforms (来自 torchvision), torch\_pruning, 以及自定义模块 model, dataset, 和 NewResNet。  2、解析配置文件：  使用 utils.read\_config() 解析 config.json 文件，获取训练所需的配置参数。  3、设置随机种子：  设置随机种子以确保结果的可重复性： utils.set\_seed(config['seed'])。  4、读取数据：  从配置文件中读取数据根目录。  使用 read\_data 方法读取训练集和验证集的图像路径和标签。  5、设置数据处理：  定义数据预处理操作，包括调整图像大小和转换为Tensor。  6、构建数据集和数据加载器：  构建训练和验证数据集 (MyDataSet) 以及相应的数据加载器 (DataLoader)，设置批次大小和工作核心数。  7、实例化模型：  使用配置文件中的参数实例化ResNet模型 (resnet18) 并设置模型运行设备。  8、保存初始模型：  将初始化后的模型参数保存到指定路径。  9、模型剪枝设置：  遍历模型层，忽略某些层（如线性层）。  定义测试输入和重要性评估方法。  初始化剪枝器 (MetaPruner)。  10、设置优化器和学习率调度器：  根据配置文件选择优化器 (Adam 或 SGD)。  设置学习率调度器 (CosineAnnealingLR)。  11、训练模型：  循环训练模型若干个epoch。  在每个epoch中：  调用 utils.trainer 进行训练，记录训练损失和准确率。  更新学习率调度器。  按照指定频率进行模型剪枝，并记录剪枝后的参数量和计算量。  按照指定频率进行模型评估，记录验证损失和准确率。  12、保存最终模型：  将最后一次训练得到的模型参数保存到指定路径。   1. 实验结果与分析（需给出结果图或表，并进行分析）   {C44B822E-C6A3-4ea7-A29C-4F7A0104F1F7}{E53FB402-590B-43f4-8879-A9042B473F56}   1. 实验的收获及心得体会（或遇到的问题，如何解决）   这次实验通过从数据集构建、模型训练、模型优化到实际部署的完整流程，让我对表情识别技术有了深入的理解和实践经验。掌握了ResNet18网络的结构和剪枝操作，积累了使用嵌入式开发板进行模型部署的经验。通过不断优化和调整，最终实现了一个高效且实用的表情识别系统。这次实验不仅提升了我的技术能力，还增强了我对AI在实际应用中潜力的信心。   1. 实验代码   Main：  import torch import utils import numpy as np import random import os from torch.utils.tensorboard import SummaryWriter import time from torchvision import transforms import torch\_pruning as tp from model import ResNet  from dataset import read\_data, MyDataSet from NewResNet import resnet18  if \_\_name\_\_ == '\_\_main\_\_':   #解析撰写的 config.json 文件：  config = utils.read\_config()   # #设置随机种子  # utils.set\_seed(config['seed'])   #以当前时间戳的形式保存 tensorboard 日志文件  t = time.localtime()  log\_path = ('logs/' + str(t.tm\_year) + '\_' + str(t.tm\_mon) + '\_' +  str(t.tm\_mday) + '\_' + str(t.tm\_hour) + '\_' + str(t.tm\_min) + '\_' + str(t.tm\_sec))  os.makedirs(log\_path)  tb\_writer = SummaryWriter(log\_dir=log\_path)   #读取训练集中所有图像的地址：  # data\_root = "D:\\个人信息\\大三下\\智能系统专业实践\\work6\\ch6\_py\\ch6\\ch6\_data"  data\_root = config['data\_root']  train\_images\_path, train\_images\_label = read\_data(root=data\_root, data\_type='train')  val\_images\_path, val\_images\_label = read\_data(root=data\_root, data\_type='val')   #读取测试集中所有图像地址的方法与上述相同，请自己撰写。   #设置批次大小与工作核心数：  batch\_size = config['batch\_size']  nw = config['num\_workers']  # 数据预处理：  data\_transform = {  "train": transforms.Compose([transforms.Resize([config['img\_size'],  config['img\_size']]), transforms.ToTensor()]),  "val": transforms.Compose([transforms.Resize([config['img\_size'],  config['img\_size']]), transforms.ToTensor()])}   # 构建训练数据集：  train\_dataset = MyDataSet(images\_path=train\_images\_path,  images\_class=train\_images\_label, transform=data\_transform["train"])  # 构建训练 dataloader：  train\_loader = torch.utils.data.DataLoader(train\_dataset, batch\_size=batch\_size, shuffle=True,  pin\_memory=True, num\_workers=nw, drop\_last=True)  # train\_loader = torch.utils.data.DataLoader(train\_dataset, batch\_size=batch\_size, shuffle=True, num\_workers=nw)   #测试的数据集构建与 dataloader 构建与之类似，请自己撰写。  test\_dataset = MyDataSet(images\_path=val\_images\_path,  images\_class=val\_images\_label, transform=data\_transform["val"])  test\_loader = torch.utils.data.DataLoader(test\_dataset, batch\_size=batch\_size, shuffle=True,  pin\_memory=True, num\_workers=nw, drop\_last=True)  #如果有验证集，也请自己撰写。   #实例化自己的训练模型：  num\_classes = config['num\_classes']  # 确定数据类型  device = config['device']  # 定义模型  train\_model = resnet18(num\_classes).to(device)  # train\_model = train\_model.cuda() # 将训练模型设置在显卡上   save\_name = config['save\_path'] + "model-{}-{}-last.pth".format(0, 0)  torch.save(train\_model.state\_dict(), save\_name)    # 加载模型超参数  # train\_model.load\_state\_dict(torch.load("D:\个人信息\大三下\智能系统专业实践\work6\ch6\_py\ch6\save\_path\\model-0-last.pth", map\_location=device))  # train\_model.eval()  # print("load model successfully")   # 模型裁剪  # 对所有 BN 逐层更新稀疏训练梯度，在 main.py 文件中模型实例化后，遍历模型，找到模型的线性层和最后的输出层，将其在剪枝中忽略。  ignored\_layers = []  for m in train\_model.modules():  if isinstance(m, torch.nn.Linear) and m.out\_features == 7:  ignored\_layers.append(m)  # 定义一个测试输入：  example\_inputs = torch.randn(1, 3, 224, 224).cuda()  # 使用上述定义的重要性评估：  imp = utils.MySlimmingImportance()  # 依据训练代数，设置迭代次数：  # 初始化剪枝器：  pruner = tp.pruner.MetaPruner(  train\_model,  example\_inputs,  importance=imp,  iterative\_steps=config['prue\_epoches'],  ch\_sparsity=0.5, # 目标稀疏性  ignored\_layers=ignored\_layers,  )   # 设置优化器：  if config['optimizer'] == 'Adam':  optimizer = torch.optim.Adam(train\_model.parameters(), lr=config['learning\_rate'],  betas=(0.9, 0.999), eps=1e-08, weight\_decay=0, amsgrad=False)  elif config['optimizer'] == 'SGD':  optimizer = torch.optim.SGD(train\_model.parameters(),  lr=config['learning\_rate'],  momentum=0.9,  dampening=0,  weight\_decay=0,  nesterov=False)  else:  raise ValueError("Optimizer must be Adam or SGD, got {} ".format(config['optimizer']))   # 撰写学习率衰减：  scheduler = torch.optim.lr\_scheduler.CosineAnnealingLR(optimizer, T\_max=config['epoches'], eta\_min=0)  # 记录每一代的训练日志，并保存在 tensorboard 文件内：  tags = ['train\_loss', 'train\_acc', 'test\_loss', 'test\_acc', 'learning\_rate', 'Params', 'MACs']  last\_test\_acc = 0  # 调用训练脚本进行训练：  for epoch in range(config['epoches']):   train\_loss, train\_acc = utils.trainer(train\_model, optimizer, train\_loader, device, epoch)  tb\_writer.add\_scalar(tags[0], train\_loss, epoch)  tb\_writer.add\_scalar(tags[1], train\_acc, epoch)  tb\_writer.add\_scalar(tags[4], optimizer.param\_groups[0]['lr'], epoch)  # 并在每一代训练完成后执行一次学习率衰减：  scheduler.step()   if epoch % config['prue\_epoches'] == 0:  if epoch != 0: # 模型裁剪  pruner.step()  macs, nparams = tp.utils.count\_ops\_and\_params(train\_model, example\_inputs)  print("Params: {:.2f} M".format(nparams / 1e6)) # 参数量  print("MACs: {:.2f} G".format(macs / 1e9)) # 计算量  tb\_writer.add\_scalar(tags[5], nparams / 1e6, epoch)  tb\_writer.add\_scalar(tags[6], macs / 1e9, epoch)   if epoch % config['test\_epoches'] == 0:  test\_loss, test\_acc = utils.evaluater(train\_model, test\_loader, device, epoch)  tb\_writer.add\_scalar(tags[2], test\_loss, epoch)  tb\_writer.add\_scalar(tags[3], test\_acc, epoch)  last\_test\_acc = test\_acc    # s最后保存最后一次训练得到的模型参数：  save\_name = config['save\_path'] + "model-{}-last.pth".format(epoch)  torch.save(train\_model.state\_dict(), save\_name)  表情识别：  import torch import utils import cv2 import numpy as np from torchvision import transforms from PIL import Image from NewResNet import resnet18  config = utils.read\_config()  # 实例化自己的网络并载入训练参数  data\_trans = transforms.Compose([transforms.Resize([224, 224]),  transforms.ToTensor()]) label\_name = ['Surprise', 'Fear', 'Disgust', 'Happiness', 'Sadness', 'Anger', 'Neutral'] device = config['device'] cap = cv2.VideoCapture(0) face\_cascade = cv2.CascadeClassifier('haarcascade\_frontalface\_default.xml')  num\_classes = config['num\_classes'] device = config['device'] model = resnet18(num\_classes).to(device) # 网络模型已经加载，请在此处加载权重路径对应的权重文件 # 加载模型超参数 model.load\_state\_dict(torch.load(config['model\_path'], map\_location=device)) model.eval() print("load model successfully")  while True:  ret, frame = cap.read()  if ret:  gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)  faces\_rects = face\_cascade.detectMultiScale(gray, scaleFactor=1.1,  minNeighbors=5)  # for (x, y, w, h) in faces\_rects:  # cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)  # face\_img = gray[y: y + h, x: x + w]  # face\_img = np.tile(face\_img, 3).reshape((w, h, 3))  # face\_img = Image.fromarray(face\_img)  # face\_img = data\_trans(face\_img)   # face\_img = face\_img.unsqueeze(0)  # face\_img = face\_img.to(device)  # np\_face\_img = face\_img.cpu().numpy()  # label\_pd = model(face\_img)  # predict\_np = np.argmax(label\_pd.cpu().detach().numpy(), axis=1)  # fer\_text = label\_name[predict\_np[0]]  # font = cv2.FONT\_HERSHEY\_SIMPLEX  # pos = (x, y)  # font\_size = 1.5  # color = (0, 0, 255)  # thickness = 2  # cv2.putText(frame, fer\_text, pos, font, font\_size, color, thickness,  # cv2.LINE\_AA)   for (x, y, w, h) in faces\_rects:  x\_lu, y\_lu = max(0, x-config['extend\_length']), max(0, y-config['extend\_length'])  height, width, \_ = frame.shape  x\_rd, y\_rd = min(width, x+config['extend\_length']+w), min(height, y+config['extend\_length']+h)  cv2.rectangle(frame, (x\_lu, y\_lu), (x\_rd, y\_rd), (0, 255, 0), 2)  face\_img = frame[y\_lu:y\_rd, x\_lu:x\_rd]  face\_img = Image.fromarray(face\_img)  face\_img = data\_trans(face\_img)  face\_img = face\_img.unsqueeze(0)  face\_img = face\_img.to(device)  label\_pd = model(face\_img)  predict\_np = np.argmax(label\_pd.cpu().detach().numpy(), axis=1)  fer\_text = label\_name[predict\_np[0]]  font = cv2.FONT\_HERSHEY\_SIMPLEX  pos = (x, y)  font\_size = 1.5  color = (0, 0, 255)  thickness = 2  cv2.putText(frame, fer\_text, pos, font, font\_size, color, thickness,  cv2.LINE\_AA)   cv2.imshow('Face Detection', frame)  if cv2.waitKey(1) & 0xFF == ord('q'):  break  # 使用 release()方法释放摄像头，并使用 destroyAllWindows()方法关闭所有窗口 cap.release() cv2.destroyAllWindows() |