

程序报告

学号: 2313211

姓名: 王众

一、问题描述

1.1 实验背景

2020年一场席卷全球的新型冠状病毒给人们带来了沉重的生命财产的损失。有效防御这种传染病毒的方法就是积极佩戴口罩。我国对此也采取了严肃的措施,在公共场合要求人们必须佩戴口罩。在本次实验中,我们要建立一个目标检测的模型,可以识别图中的人是否佩戴了口罩。

1.2 实验要求

- 建立深度学习模型,检测出图中的人是否佩戴了口罩,并将其尽可能调整到最佳状态。
- 学习经典的模型 mtcnn 和 mobilenet 的结构。
- 学习训练时的方法。

1.3 实验环境

可以使用基于 Python 的 OpenCV、PIL 库进行图像相关处理,使用 Numpy 库进行相关数值运算,使用 **MindSpore**、**Keras** 等框架建立深度学习模型等。

1.4 注意事项

- Python 与 Python Package 的使用方式,可在右侧 **API文档** 中查阅。
- 当右上角的『Python 3』长时间指示为运行中的时候,造成代码无法执行时,可以重新启动 Kernel 解决(左上角『Kernel』-『Restart Kernel』)。

1.5 参考资料

- 论文 Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks: https://kpzhang93.github.io/MTCNN_face_detection_alignment/
- OpenCV: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_tutorials.html
- PIL: <https://pillow.readthedocs.io/en/stable/>
- Numpy: <https://www.numpy.org/>
- Scikit-learn: <https://scikit-learn.org/>
- PyTorch: <https://pytorch.org/>

二、设计思想

本实验我们采用的是搭建并训练 *MobileNet* 模型来进行深度学习,从而实现目标检测的任务。在实验中我们通过训练 *MobileNet* 网络,调整其参数来实现模型的优化,达到更准确的检测图中的人是否佩戴口罩的目的。

本次实验我们的目标是,通过调节几个关键的参数,来使我们的损失值降到最低,来获取最后的测试分数。我们下面就列举一下几个重要的参数:

```
epochs = 50
model = MobileNetV1(classes=2).to(device)
optimizer = optim.Adam(model.parameters(), lr=1e-3) # 优化器
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer,
                                                    'max',
                                                    factor=0.5,
                                                    patience=2)
```

可以看到，我们的所有关键的参数都在上面了，我们主要去调节的是`epochs`(就是训练的总轮数)；`lr`；`factor`和`patience`

我们先对原来的模型进行测试，得到的结果是60分，太低了，说明我们的参数很不合理。

我们进入我们的训练日志中，发现我们的损失值是先减少后增加，而且越到后面变化的越明显，说明我们需要调节我们的`factor`和`patience`，我们一次一次的尝试，最后发现当这两个参数分别为0.5和2时，刚好可以达到目前的最高分(90分)

然后我们继续进行调节，我们发现，当我们训练的时候，一般来说最小的损失值都在第35-40轮左右，所以我们对`epochs`进行调节，最后发现当轮数来到40轮的时候，我们获得了最小的损失值：0.0238

然后我们对其进行测试，就得到了结果，我们将其生成模型`temp_new.pth`，存储到了我们的`results`文件夹中。

最后的代码内容为：

```
epochs = 40
model = MobileNetV1(classes=2).to(device)
optimizer = optim.Adam(model.parameters(), lr=1e-3) # 优化器
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer,
                                                    'max',
                                                    factor=0.2,
                                                    patience=12)
```

三、代码内容

3.1 train.py

```
import warnings
# 忽视警告
warnings.filterwarnings('ignore')

import cv2
from PIL import Image
import numpy as np
import copy
import matplotlib.pyplot as plt
from tqdm.auto import tqdm
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision.datasets import ImageFolder
```

```

import torchvision.transforms as T
from torch.utils.data import DataLoader

from torch_py.Utills import plot_image
from torch_py.MTCNN.detector import FaceDetector
from torch_py.MobileNetV1 import MobileNetV1
from torch_py.FaceRec import Recognition

from torch_py.Utills import plot_image
from torch_py.MTCNN.detector import FaceDetector
from torch_py.MobileNetV1 import MobileNetV1
from torch_py.FaceRec import Recognition
from torch_py.FaceRec import Recognition
from PIL import Image
import cv2

def processing_data(data_path, height=224, width=224, batch_size=32,
                    test_split=0.1):
    """
    数据处理部分
    :param data_path: 数据路径
    :param height: 高度
    :param width: 宽度
    :param batch_size: 每次读取图片的数量
    :param test_split: 测试集划分比例
    :return:
    """
    transforms = T.Compose([
        T.Resize((height, width)),
        T.RandomHorizontalFlip(0.1), # 进行随机水平翻转
        T.RandomVerticalFlip(0.1), # 进行随机竖直翻转
        T.ToTensor(), # 转化为张量
        T.Normalize([0], [1]), # 归一化
    ])

    dataset = ImageFolder(data_path, transform=transforms)
    # 划分数据集
    train_size = int((1-test_split)*len(dataset))
    test_size = len(dataset) - train_size
    train_dataset, test_dataset = torch.utils.data.random_split(dataset,
    [train_size, test_size])
    # 创建一个 DataLoader 对象
    train_data_loader = DataLoader(train_dataset,
    batch_size=batch_size,shuffle=True)
    valid_data_loader = DataLoader(test_dataset,
    batch_size=batch_size,shuffle=True)

    return train_data_loader, valid_data_loader

data_path = './datasets/5f680a696ec9b83bb0037081-momodel/data/image'

# 加载 MobileNet 的预训练模型权
device = torch.device("cuda:0") if torch.cuda.is_available() else
torch.device("cpu")
train_data_loader, valid_data_loader = processing_data(data_path=data_path,
height=160, width=160, batch_size=32)

```

```

epochs = 40
model = MobileNetV1(classes=2).to(device)
optimizer = optim.Adam(model.parameters(), lr=1e-3) # 优化器
# 学习率下降的方式, acc三次不下降就下降学习率继续训练, 衰减学习率
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer,
                                                    'max',
                                                    factor=0.2,
                                                    patience=12)

# 损失函数
criterion = nn.CrossEntropyLoss()
best_loss = 1e9
best_model_weights = copy.deepcopy(model.state_dict())
loss_list = [] # 存储损失函数值
for epoch in range(epochs):
    model.train()

    for batch_idx, (x, y) in tqdm(enumerate(train_data_loader, 1)):
        x = x.to(device)
        y = y.to(device)
        pred_y = model(x)

        # print(pred_y.shape)
        # print(y.shape)

        loss = criterion(pred_y, y)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if loss < best_loss:
            best_model_weights = copy.deepcopy(model.state_dict())
            best_loss = loss

    loss_list.append(loss)

    print('step:' + str(epoch + 1) + '/' + str(epochs) + ' || Total Loss: %.4f' %
          (loss))
    torch.save(model.state_dict(), './results/temp.pth')
print('Finish Training.')

```

3.2main.py

```

import warnings
# 忽视警告
warnings.filterwarnings('ignore')

import cv2
from PIL import Image
import numpy as np
import copy
import matplotlib.pyplot as plt
from tqdm.auto import tqdm

```

```

import torch
import torch.nn as nn
import torch.optim as optim
from torchvision.datasets import ImageFolder
import torchvision.transforms as T
from torch.utils.data import DataLoader

from torch_py.Utills import plot_image
from torch_py.MTCNN.detector import FaceDetector
from torch_py.MobileNetV1 import MobileNetV1
from torch_py.FaceRec import Recognition

from torch_py.Utills import plot_image
from torch_py.MTCNN.detector import FaceDetector
from torch_py.MobileNetV1 import MobileNetV1
from torch_py.FaceRec import Recognition
from torch_py.FaceRec import Recognition
from PIL import Image
import cv2

# ----- 请加载您最满意的模型 -----
# 加载模型(请加载你认为的最佳模型)
# 加载模型,加载请注意 model_path 是相对路径, 与当前文件同级。
# 如果你的模型是在 results 文件夹下的 dnn.h5 模型, 则 model_path = 'results/temp.pth'
model_path = 'results/temp_new.pth'
# -----

def predict(img):
    """
    加载模型和模型预测
    :param img: cv2.imread 图像
    :return: 预测的图片中的总人数、其中佩戴口罩的人数
    """
    # ----- 实现模型预测部分的代码 -----
    # 将 cv2.imread 图像转化为 PIL.Image 图像, 用来兼容测试输入的 cv2 读取的图像 (勿删!!!)
    # cv2.imread 读取图像的类型是 numpy.ndarray
    # PIL.Image.open 读取图像的类型是 PIL.JpegImagePlugin.JpegImageFile
    if isinstance(img, np.ndarray):
        # 转化为 PIL.JpegImagePlugin.JpegImageFile 类型
        img = Image.fromarray(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))

    recognize = Recognition(model_path)
    img, all_num, mask_num = recognize.mask_recognize(img)
    # -----
    return all_num, mask_num







```

四、实验结果

测试详情

测试点	状态	时长	结果
在 5 张图片上测试模型	✓	5s	得分:95.0

确定

Name	Last Modified
 tb_results	3 months ago
 _README.md	3 months ago
 temp_best.pth	5 days ago
 temp_best1.pth	5 days ago
 temp_best2.pth	5 days ago
 temp.pth	5 days ago

在经过二十多轮的训练之后，我们的正确率始终只能在95分，但并不能上升到最终的一百分。在峰值之后会直接回到90分。我们将所有得分为95分的模型均进行了保存并以`tmp_best.pth`进行命名。

五、心得体会

我们通过该实验，学习到了调参的技术：如何调参？什么时候需要调参？往什么方向调参？等等。我们从刚开始的60分，到最后调试出来的95分，经历了二十多次次调参的过程，在一步一步靠近答案的过程中，也享受到了调参的乐趣。

当然，我们也有很多的改进方向，比如说我们可以更换其他的模型，或者换取其他类型的损失函数如交叉熵损失函数等来进行模型的训练，在数据处理方面采用更多的处理方法，如随机裁取等，便于提取出不同的图片特征以更好地对网络模型进行训练。

此次项目代码部分不难，但是重点在于调参的过程，通过这次实验，我对机器学习、深度学习有了更深的了解，对于模型的调用与训练也更加熟练了。