MMEdu

mmedu

2022 年 08 月 19 日

关于 MMEdu

MMEdu 帮助文档

更多资讯请转至 XEdu 文档。

CHAPTER 1

MMEdu 的故事

1.1 1.MMEdu 的缘起

MMEdu 的缘起有很多版本。

版本一

上海人工智能实验室成立初期，智能中心负责人戴娟找上海交大谈国智班的相关事宜。

……

版本二

2020 年，清华大学出版社的义务教育阶段信息技术教材开始修订，其中九年级分册要涉及人工智能。经过一番调研，几位主编发现小学教材可以用 Mind+，高中教材绝大多数使用了 Keras，那么初中呢？似乎只能选择 Keras。

……

1.2 2. 为什么要开发 MMEdu

中小学的 ai 教育需要低门槛的框架或者工具。最常用是 keras，存在如下问题：

1）代码过于底层，如搭建神经网络

2）离应用太遥远，解决不了真实问题

3）有些直接可用的，又封装过度，如 OpenCV；

4）更换多个工具，门槛太高。如物体检测的 Yolov3，因为基于 PyTorch，采用项目式，基本上教师无法使用，再如图片风格化（学习迁移，对抗网络）

1.3 3. 对 MMEdu 的期望

1）用最好的、最先进的 AI 工具

专注于计算机视觉识别，将 OpenMMLab 这一高校研究工具降维到中小学课堂。

2）开箱即用的 AI 工具

既支持底层的网络搭建，又引入最新的 SOTA 模型，代码简洁，便于理解。一套工具，完成神经网络和视觉学习。

3）真正的 AI 开发工具

基于 PyTorch，能训练出近工业级的模型，能够解决真实问题。

CHAPTER 2

开发和维护团队

2.1 1. 项目策划

1）戴娟

硕士（人工智能方向），上海人工智能实验室智能教育中心主任，项目总负责。

2）谢作如

高中信息技术正高级教师，浙江省特级教师，负责具体研发进度和课程开发。

3）张崇珍

硕士（计算机视觉方向），上海人工智能实验室智能教育中心项目经理，负责项目的外部沟通和协调。

4）

2.2 2. 算法团队

1）贾彦灏（中国科学院大学）

2）王博伦（上海交通大学）

3）高剑雄（上海复旦大学）

4）丛培珊（上海科技大学）

5）姜燕炜（上海交通大学）

6）张卉婧（上海复旦大学）

7）徐泽庭（华东师范大学）

……

2.3 3. 课程团队

1）陆雅楠（上海师范大学）
2）邱奕盛（华东师范大学）
3）周子皓（华东师范大学）
4）胡潇桓（美国纽约大学）
5）……

2.4 4. 测试团队

CHAPTER 3

版本更新记录

3.1 1. 正式版

发布时间：将在 2022 年 9 月发布
版本说明：

3.2 2. 测试版

3.2.1 1）0.5 版

发布时间：2022.4
版本说明：整合图像分类（cls）、物体检测（det）两个核心模块，内置 Pyzo、Jupyter，实现一键部署。

3.2.2 2）0.7 版

发布时间：2022.6
版本说明：优化 0.5 版两个模块，新增自定义网络（MMBase）模块。

CHAPTER 4

MMEdu 大事记

2022 年 6 月，发布 0.7 版。

2022 年 4 月，发布 0.5 版。

2022 年 3 月，完成内测版，确定目录，规范路径；同步编写课程，设计 AI 实验。

2022 年 2 月，确定语法风格。

2022 年 1 月，工作启动，组建核心团队。

2021 年 11 月，确定想法——将 OpenMMLab “降维”，让中小学生能够使用。

CHAPTER 5

MMEdu 简介

5.1 1.MMEdu 是什么？

MMEdu 源于国产人工智能视觉（CV）算法集成框架 OpenMMLab，是一个“开箱即用”的深度学习开发工具。在继承 OpenMMLab 强大功能的同时，MMEdu 简化了神经网络模型搭建和训练的参数，降低了编程的难度，并实现一键部署编程环境，让初学者通过简洁的代码完成各种 SOTA 模型（state-of-the-art，指在该项研究任务中目前最好/最先进的模型）的训练，并能够快速搭建出 AI 应用系统。

GitHub：https://github.com/OpenXLab-Edu/OpenMMLab-Edu

国内镜像：https://gitee.com/openxlab-edu/OpenMMLab-Edu

5.2 2.MMEdu 和常见 AI 框架的比较

5.2.1 1）MMEdu 和 OpenCV 的比较

OpenCV 是一个开源的计算机视觉框架，MMEdu 的核心模块 MMCV 基于 OpenCV，二者联系紧密。

OpenCV 虽然是一个很常用的工具，但是普通用户很难在 OpenCV 的基础上训练自己的分类器。MMEdu 则是一个入门门槛很低的深度学习开发工具，借助 MMEdu 和经典的网络模型，只要拥有一定数量的数据，连小学生都能训练出自己的个性化模型。

5.2.2 2）MMEdu 和 MediaPipe 的比较

MediaPipe 是一款由 Google Research 开发并开源的多媒体机器学习模型应用框架，支持人脸识别、手势识别和表情识别等，功能非常强大。MMEdu 中的 MMPose 模块关注的重点也是手势识别，功能类似。但 MediaPipe 是应用框架，而不是开发框架。换句话说，用 MediaPipe 只能完成其提供的 AI 识别功能，没办法训练自己的个性化模型。

5.2.3 3）MMEdu 和 Keras 的比较

Keras 是一个高层神经网络 API，是对 Tensorflow、Theano 以及 CNTK 的进一步封装。OpenMMLab 和 Keras 一样，都是为支持快速实验而生。MMEdu 则源于 OpenMMLab，其语法设计借鉴过 Keras。

相当而言，MMEdu 的语法比 Keras 更加简洁，对中小学生来说也更友好。目前 MMEdu 的底层框架是 Pytorch，而 Keras 的底层是 TensorFlow（虽然也有基于 Pytorch 的 Keras）。

5.2.4 4）MMEdu 和 FastAI 的比较

FastAI（Fast.ai）最受学生欢迎的 MOOC 课程平台，也是一个 PyTorch 的顶层框架。和 OpenMMLab 的做法一样，为了让新手快速实施深度学习，FastAI 团队将知名的 SOTA 模型封装好供学习者使用。

FastAI 同样基于 Pytorch，但是和 OpenMMLab 不同的是，FastAI 只能支持 GPU。考虑到中小学的基础教育中很难拥有 GPU 环境，MMEdu 特意将 OpenMMLab 中支持 CPU 训练的工具筛选出来，供中小学生使用。

CHAPTER 6

MMEdu 安装

6.1 1. 使用一键安装包

为方便中小学教学，MMEdu 团队提供了一键安装包。只要下载并解压 MMEdu 的 Project 文件，即可直接使用。

第一步：下载 MMEdu 最新版文件，并解压到本地，文件夹目录结构如下图所示。

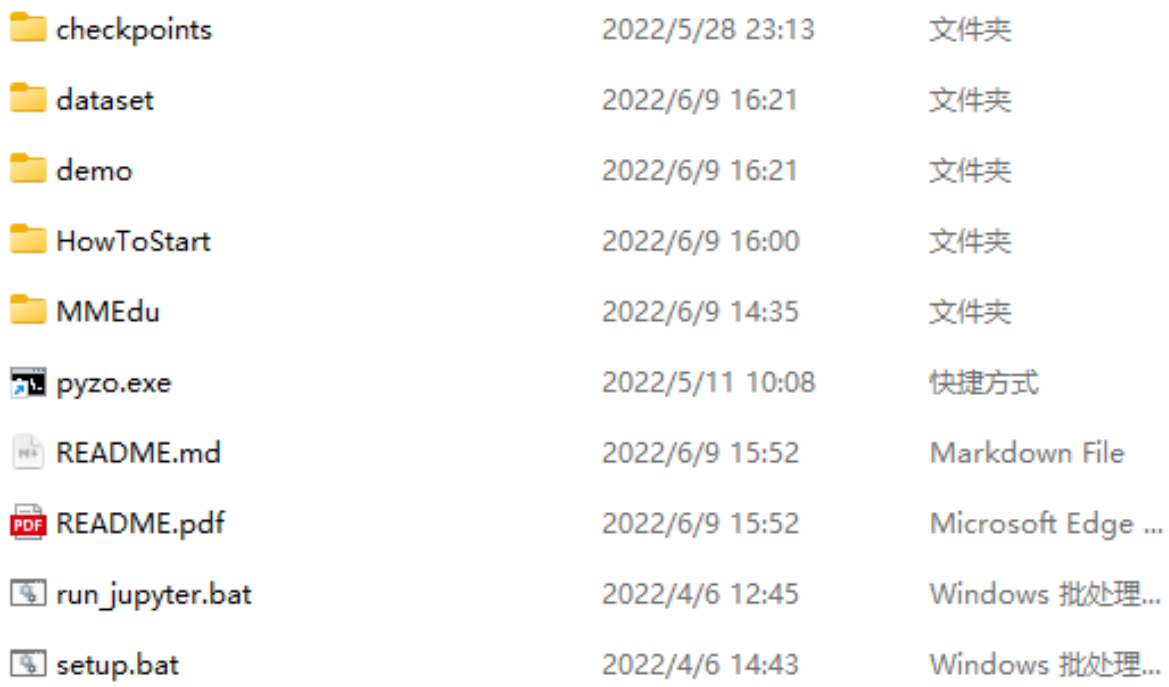


图 1 目录结构图

1）下载方式一

飞书网盘：https://p6bm2if73b.feishu.cn/drive/folder/fldcnfDtXSQx0PDuUGLWZlnVO3g

2）下载方式二

百度网盘：https://pan.baidu.com/s/19lu12-T2GF_PiI3hMbtX-g?pwd=2022

提取码：2022

第二步：运行根目录的“steup.bat”文件，完成环境部署（如下图所示）。

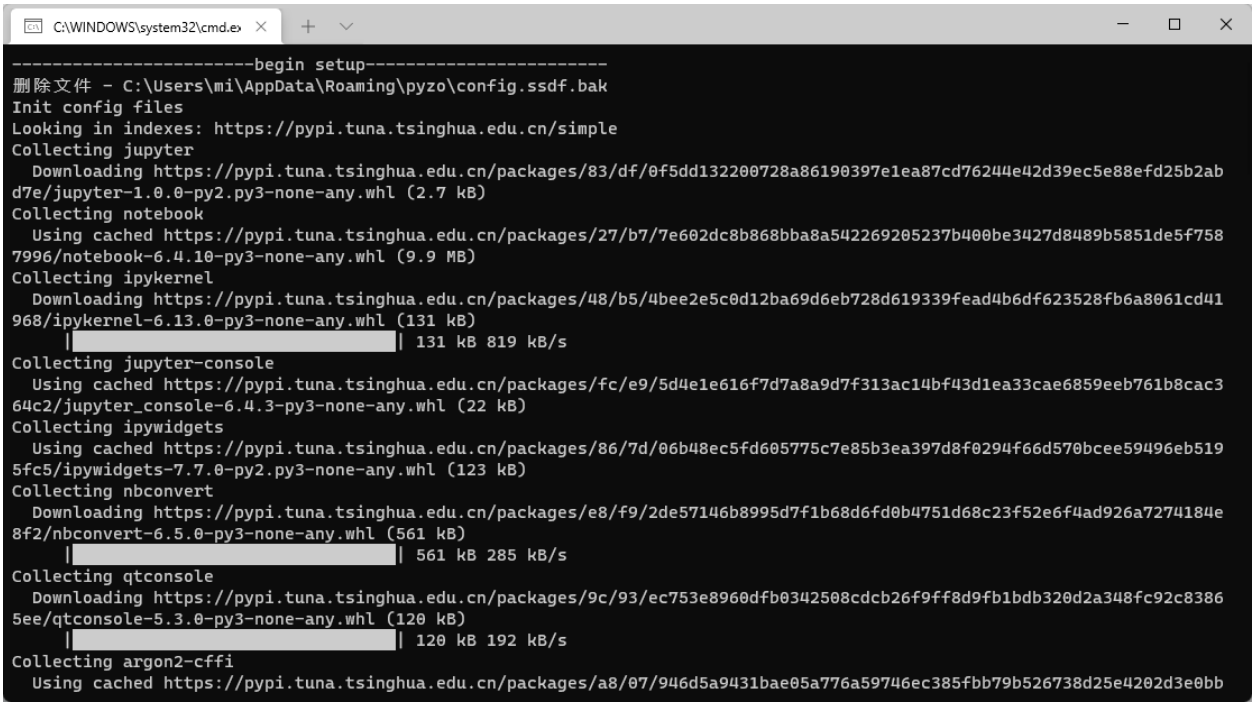


图 2 环境部署界面

第三步：您可以根据个人喜好，选择自己习惯的 IDE。

1）使用 MMEdu 自带的 Pyzo。

Pyzo 是一款好用的 Python 轻量级 IDE。其最突出的两个特点便是是简洁性和交互性。打开根目录下的 pyzo.exe 文件快捷方式即可打开 pyzo，其指向“Tools”中的“pyzo.exe”。使用 Pyzo 打开“demo”文件夹中的 py 文件，如“cls_demo.py”，点击“运行”的“将文件作为脚本运行”即可运行代码，界面如下图所示。

图 3 Pyzo 界面图

2）使用 MMEdu 自带的 Jupyter。

Jupyter Notebook 是基于网页的用于交互计算的应用程序。其可被应用于全过程计算：开发、文档编写、运行代码和展示结果。它相对简单，对用户也更加友好，适合初学者。打开根目录下的“run_jupyter.bat”，即自动启动浏览器并显示界面，如下图所示。

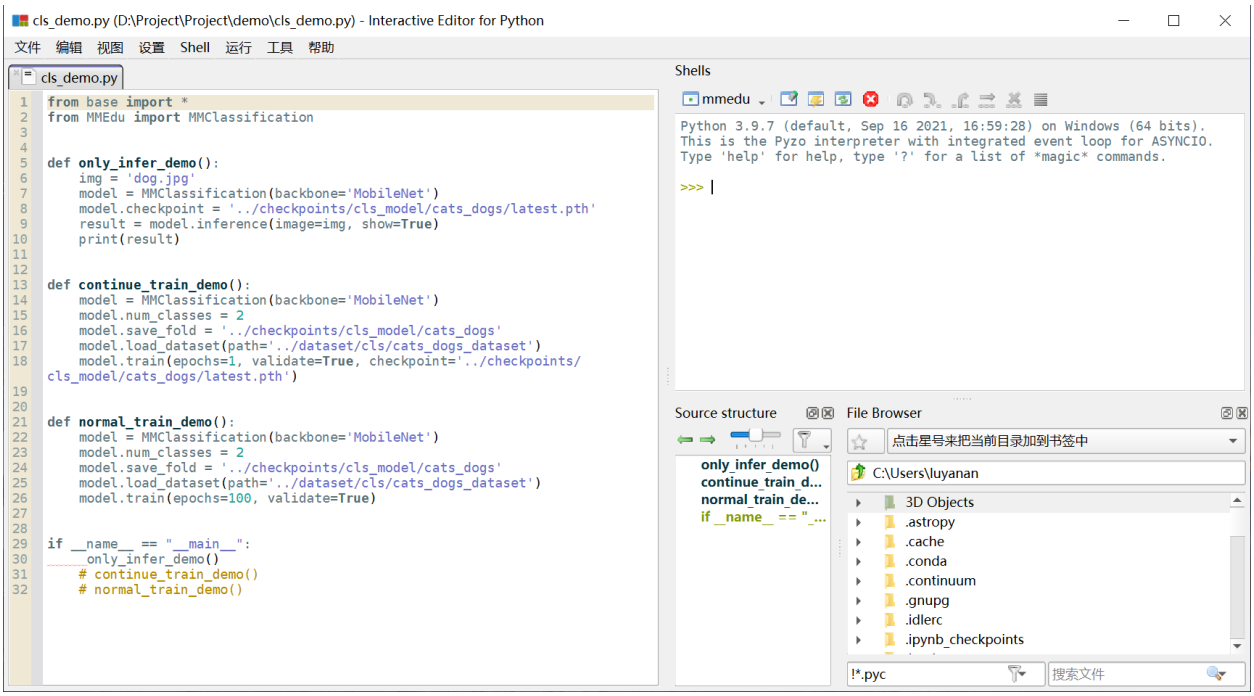


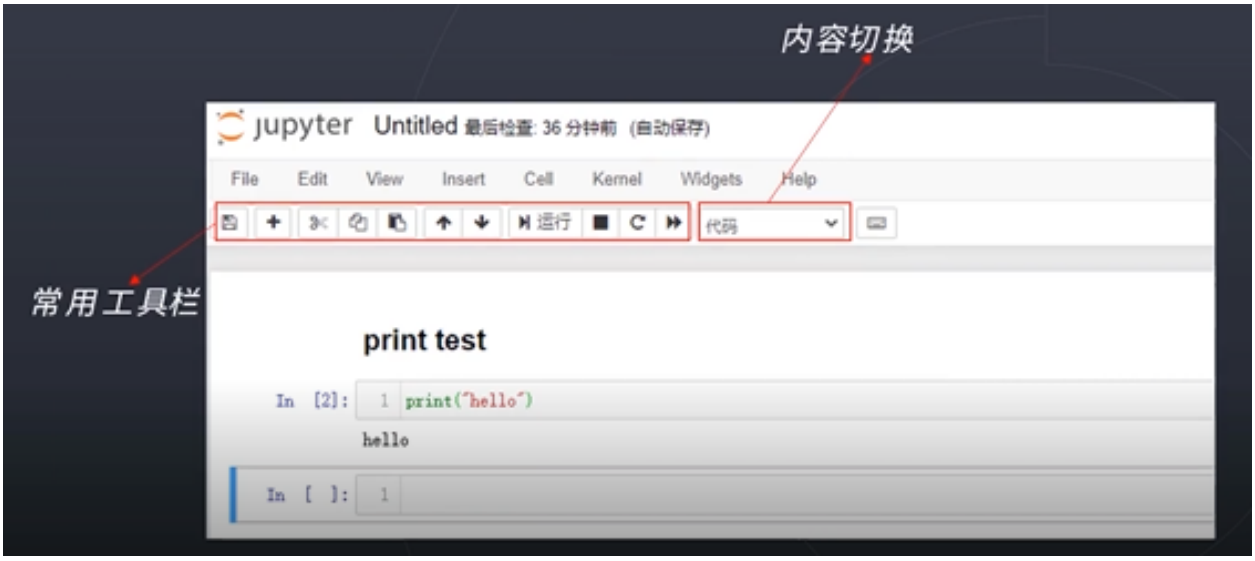
图 4 jupyter 界面图

使用常用工具栏对代码进行操作，如“运行”，可以在单元格中编写文本或者代码（如下图中写了 `print("hello")` 代码的位置），执行代码的结果也将会在每个单元下方呈现。可以逐个运行单元格，每点击一次，仅运行一个单元格。单元格左侧 [*] 内的星号变为数字，表示该单元格运行完成。此时可打开“demo”文件夹中的 ipynb 文件，如“cls_notebook.ipynb”。

图 5 jupyter 运行界面

3）使用其他 IDE。

如果您需要使用其他 IDE，那么需要您自己配置 Python 编译器，配置方法如下。



- 配置环境路径

① 打开您的 IDE，如 PyCharm、Thonny 等。

② 配置 Python 编译器，路径为解压路径下的“MMEdu”文件夹下的“mmedu”文件夹中的“python.exe”文件。PyCharm 环境路径配置如下图所示。

图 6 PyCharm 的环境路径配置界面

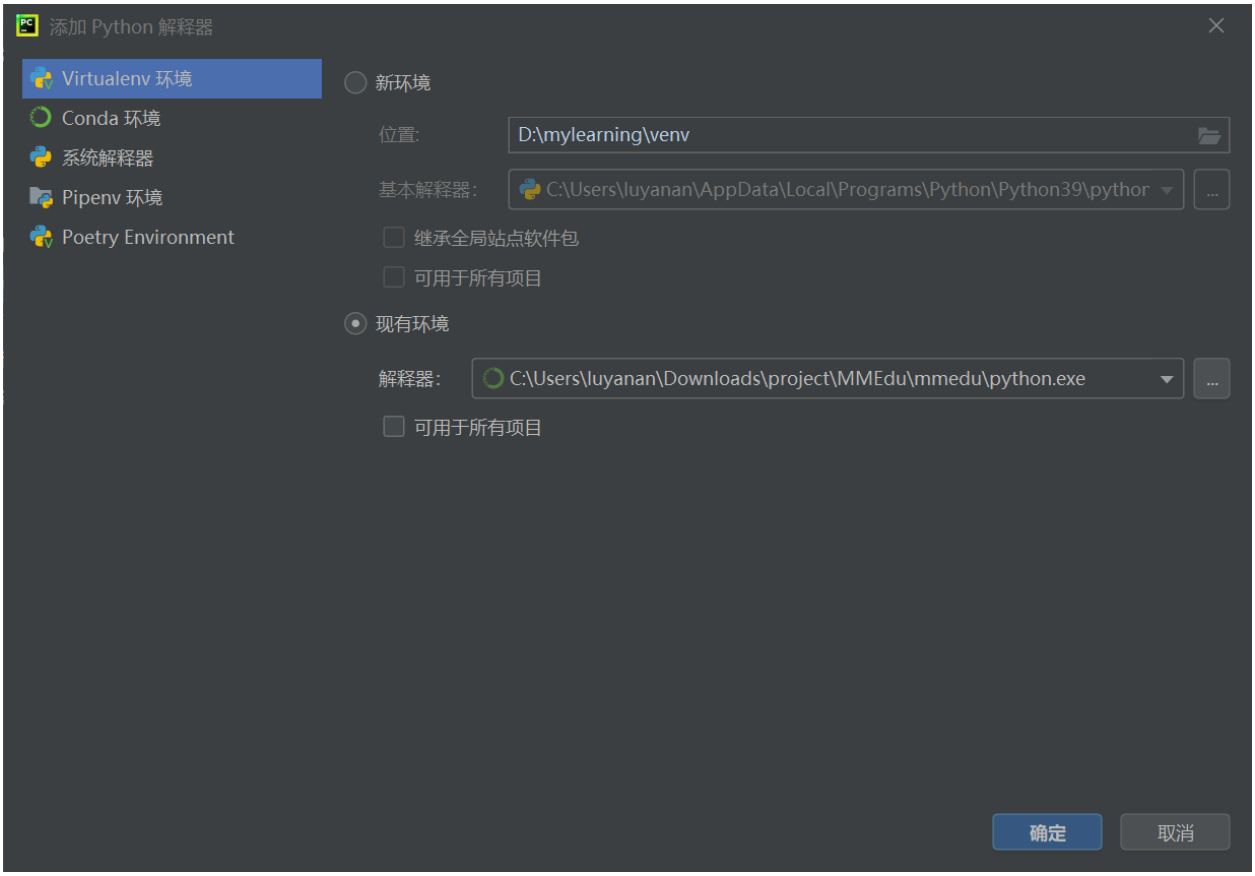
- 执行 demo 文件

用 IDE 打开解压路径下的 py 文件，如“cls_demo.py”，点击“运行”。运行效果应和 pyzo 一样。

6.2 2. 使用 Git 工具手动部署

6.2.1 1）安装须知

```
python==3.9.7
numpy==1.22.2
torch==1.8.1+cu101 # cpu环境只需保证torch==1.8.1
torchvision==0.9.1
torchaudio==0.8.1
mmcv-full==1.4.5
mmcls=0.20.1
mmdet==2.21.0
mmpose==0.23.0
```


添加 Python 解释器
Virtualenv 环境
Conda 环境
系统解释器
Pipenv 环境
Poetry Environment
新环境
位置:
D:\mylearning\venv
基本解释器:
C:\Users\luyanan\AppData\Local\Programs\Python\Python39\pythor
继承全局站点软件包
可用于所有项目
现有环境
解释器:
C:\Users\luyanan\Downloads\project\MMEdu\mmedu\python.exe
可用于所有项目
确定
取消

6.2.2 2）安装环境

- **安装 python**

下载 python，版本号 3.9.7，勾选“Add Python 3.9 to PATH”将 Python 编辑器加入系统环境变量，完成安装。

- **安装 anaconda**

anaconda 是一个 python 的发行版，包括了 python 和很多常见的软件库, 和一个包管理器 conda，我们后续可以在 conda 中创建一个新的虚拟环境来方便我们后续的包管理。可在清华 anaconda 镜像源下载 anaconda 的安装包进行安装。

安装时注意勾选“Add Anaconda to the system PATH environment variable”将安装路径自动添加系统环境变量和“Register Anaconda as the system Python3.9”默认使用 python 的版本。

- **配置虚拟环境**

打开 anaconda prompt 后输入：

```
conda create -n openmmlab python==3.9.7
```

输入 yes 之后等待安装完成，然后激活你的新环境开始最后的配置：

```
conda activate openmmlab
```

6.2.3 3）安装相关依赖库

可以使用清华源进行 pip 安装（可以选择在本地安装，也可以激活虚拟环境在虚拟环境中安装）。

- **安装 PyTorch 和 torchvision**

```
pip install torch==1.8.1 torchvision==0.9.1 -i https://pypi.tuna.tsinghua.edu.cn/
↪simple
```

- **安装 mmcv-full**

```
pip install mmcv-full==1.4.5 -i https://pypi.tuna.tsinghua.edu.cn/simple
```

注：如果您没有 GPU 加速计算，那么就安装普通版本的 mmcv：

```
pip install mmcv
```

- **安装 numpy**

```
pip install numpy==1.22.2 -i https://pypi.tuna.tsinghua.edu.cn/simple
```

6.2.4 4）安装 mm 模块

可以从 GitHub 上克隆 mm 模块的代码库并安装（同安装相关依赖库，可以选择两种方式安装）。

- **安装 git**

下载链接：https://git-scm.com/download/win

- **安装 mmcls**

新建文件夹，克隆 mmclassification 代码库。

```
git clone --branch v0.21.0 http://github.com/open-mmlab/mmclassification.git
```

复制克隆下来的代码文件夹路径，使用 cd 命令切换到文件路径。

```
cd mmclassification
```

安装。

```
pip install -e .
```

- **安装 mmdet**

新建文件夹，克隆 mmdetection 代码库。

```
git clone --branch v2.21.0 https://github.com/open-mmlab/mmdetection.git
```

复制克隆下来的代码文件夹路径，使用 cd 命令切换到文件路径。

```
cd mmdetection
```

安装。

```
pip install -v -e .
```

注：若遇“ERROR: Failed cleaning build dir for pycocotools”

解决方式：安装 visual studio installer（版本高于 2015 即可）

下载地址：https://visualstudio.microsoft.com/zh-hans/thank-you-downloading-visual-studio/?sku=Community&channel=Release&version=VS2022&source=VSLandingPage&cid=2030&passive=false

安装时勾选工作负荷-桌面应用和移动应用-使用 C++ 的桌面开发，安装完成后再次启动 mmdet 安装。

- **安装其他模块**

可以使用一样的安装方法安装其他模块。

6.2.5 5）查看已安装的模块

可通过查看已安装的模块，确认是否安装成功，可使用如下命令。

```
pip list
```

图 7 安装列表

```
addict              2.4.0
aiohttp             3.7.4.post0
async-timeout       3.0.1
attrs               21.4.0
certifi             2021.10.8
cffi                1.15.0
chardet             4.0.0
cycler              0.11.0
discord.py          1.7.3
fonttools           4.32.0
idna                3.3
kiwisolver          1.4.2
matplotlib          3.5.1
mmcls               0.21.0      d:\mmedutest\mmclassification
mmcv-full           1.4.5
mmdet               2.21.0      d:\mmedutest\mmdetection
multidict           6.0.2
numpy               1.22.3
opencv-python       4.5.5.64
packaging           21.3
Pillow              9.1.0
pip                 22.0.4
pycocotools         2.0.4
pycparser           2.21
PyNaCl              1.4.0
pyparsing           3.0.8
python-dateutil     2.8.2
PyYAML              6.0
regex               2022.3.15
setuptools          61.2.0
six                 1.16.0
terminaltables      3.1.10
torch               1.8.1
torchvision         0.9.1
typing_extensions   4.2.0
wheel               0.37.1
wincertstore        0.2
yapf                0.32.0
yarl                1.7.2
```

CHAPTER 7

体验 MMEdu

7.1 1. 运行 Demo 代码

1）使用默认 IDE

双击 pyzo.exe，打开 demo 文件夹中的 cls_demo.py，运行并体验相关功能，也可以查看其他的 Demo 文件。详细说明可以在 HowToStart 文件夹看到。

2）使用第三方 IDE

环境支持任意的 Python 编辑器，如：Thonny、PyCharm、Sublime 等。只要配置其 Python 解释器地址为 {你的安装目录}+{\MMEdu\mmedu\python.exe}。

7.2 2. 体验快速入门课程

体验入门 Demo 后，我们还准备了一系列的入门课程供您参考。将在稍晚发布。

7.3 3.MMEdu 的基本代码风格

推理：

```
from MMEdu import MMClassification as cls
img = './img.png'
model = cls(backbone='LeNet')
checkpoint = './latest.pth'
class_path = './classes.txt'
result = model.inference(image=img, show=True, class_path=class_path,checkpoint =␣
↪checkpoint)
model.print_result(result)
```

典型训练：

```
from MMEdu import MMClassification as cls
model = cls(backbone='LeNet')
model.num_classes = 3
model.load_dataset(path='./dataset')
model.save_fold = './my_model'
model.train(epochs=10, validate=True)
```

继续训练：

```
from MMEdu import MMClassification as cls
model = cls(backbone='LeNet')
model.num_classes = 3
model.load_dataset(path='./dataset')
model.save_fold = './my_model'
checkpoint = './latest.pth'
model.train(epochs=10, validate=True, checkpoint=checkpoint)
```

CHAPTER 8

训练 AI 模型

CHAPTER 9

部署 AI 应用

9.1 1. 准备工作

所谓准备工作就是先训练好一个效果不错的模型。

9.2 2. 借助 OpenCV 识别摄像头画面

1）代码编写

```
import cv2
from time import sleep
cap = cv2.VideoCapture(0)
print("一秒钟后开始拍照......")
sleep(1)
ret, frame = cap.read()
cv2.imshow("my_hand.jpg", frame)
cv2.waitKey(1000) # 显示1秒（这里单位是毫秒）
cv2.destroyAllWindows()
cv2.imwrite("my_hand.jpg", frame)
print("成功保存 my_hand.jpg")
cap.release()
```

2）运行效果

```
一秒钟后开始拍照......
成功保存 drawing.jpg
```

9.3 3. 借助 PyWebIO 部署 Web 应用

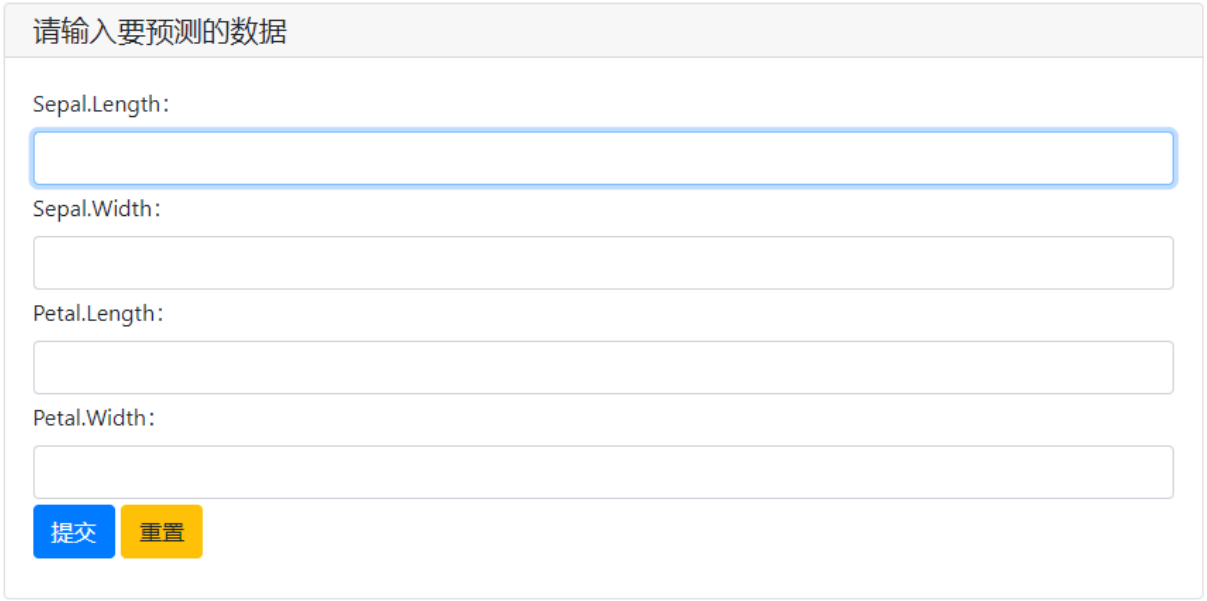
1）编写代码

```
from base import *
from MMEdu import MMBase
import numpy as np
from pywebio.input import input, FLOAT,input_group
from pywebio.output import put_text

# 鸢尾花的分类
flower = ['iris-setosa','iris-versicolor','iris-virginica']

# 声明模型
model = MMBase()
# 导入模型
model.load('./checkpoints/mmbase_net.pkl')
info=input_group('请输入要预测的数据', [
    input('Sepal.Length: ', name='x1', type=FLOAT),
    input('Sepal.Width: ', name='x2', type=FLOAT),
    input('Petal.Length: ', name='x3', type=FLOAT),
    input('Petal.Width: ', name='x4', type=FLOAT)
])
print(info)
x = list(info.values())
put_text('你输入的数据是: %s' % (x))
model.inference([x])
r=model.print_result()
put_text('模型预测的结果是: ' + flower[r[0]['预测值']])
print('模型预测的结果是: ' +flower[r[0]['预测值']])
```

2）运行效果



9.4 4. 连接开源硬件开发智能作品

1）

CHAPTER 10

MMEdu 的目录详解

MMEdu 一键安装版是一个压缩包，解压后即可使用。

MMEdu 的根目录结构如下：

```
OpenMMLab-Edu
├── MMEdu
├── checkpoints
├── dataset
├── demo
├── HowToStart
├── tools (github)
├── visualization (github)
├── base.py
├── README.md
├── README.pdf
├── setup.bat
├── 安装文件说明.ipynb
├── pyzo.exe
├── run_jupyter.bat
```

接下来对每层子目录进行介绍。

10.1 MMEdu 目录：

存放各个模块的底层代码、算法模型文件夹“models”和封装环境文件夹“mmedu”。“models”文件夹中提供了各个模块常见的网络模型，内置模型配置文件和说明文档，说明文档提供了模型简介、特点、预训练模型下载链接和适用领域等。“mmedu”文件夹打包了 MMEdu 各模块运行所需的环境和中小学课程常用的库。

10.2 checkpoints 目录：

存放各个模块的预训练模型的权重文件，分别放在以模块名称命名的文件夹下，如“cls_model”。

10.3 dataset 目录：

存放为各个模块任务准备的数据集，分别放在以模块名称命名的文件夹下，如“cls”。同时 github 上此目录下还存放了各个模块自定义数据集的说明文档，如“pose-dataset.md”，文档提供了每个模块对应的数据集格式、下载链接、使用说明、自制数据集流程。

10.4 demo 目录：

存放各个模块的测试程序，如“cls_demo.py”，并提供了测试图片。测试程序包括 `py` 文件和 `ipynb` 文件，可支持各种“Python IDE”和“jupyter notebook”运行，可运行根目录的“pyzo.exe”和“run_jupyter.bat”后打开测试程序。

10.5 HowToStart 目录：

存放各个模块的使用教程文档，如“MMClassfication 使用教程.md”，文档提供了代码详细说明、参数说明与使用等。同时 github 上此目录下还存放了 OpenMMLab 各个模块的开发文档供感兴趣的老师和同学参考，如“OpenMMLab_MMClassification.md”，提供了模块介绍、不同函数使用、深度魔改、添加网络等。

10.6 tools 目录：

存放数据集格式的转换、不同框架的部署等通用工具。后续会陆续开发数据集查看工具、数据集标注工具等工具。

10.7 visualization 目录：

存放可视化界面。

CHAPTER 11

MMEdu 的模块概述

MMEdu 基于 OpenMMLab 的基础上开发，因为面向中小学，优先选择支持 CPU 训练的模块。

模块名称	简称	功能
MMClassification	MMCLS	图片分类
MMDetection	MMDET	图片中的物体检测
MMGeneration	MMGEN	GAN，风格化
MMPose	MMPOSE	骨架
MMEditing		
MMSegmentation		像素级识别

CHAPTER 12

图像分类模块：MMClassification

12.1 简介

MMClassifiation 的主要功能是对图像进行分类。其支持的 SOTA 模型有 LeNet、MobileNet 等。

12.2 使用说明

MMEdu 中预置了“石头剪刀布”三分类的数据集，并且已经预训练了权重（路径：/checkpoints/cls_model/hand_gray/latest.pth）。在 demo 文件夹中，还提供了一张测试图片。

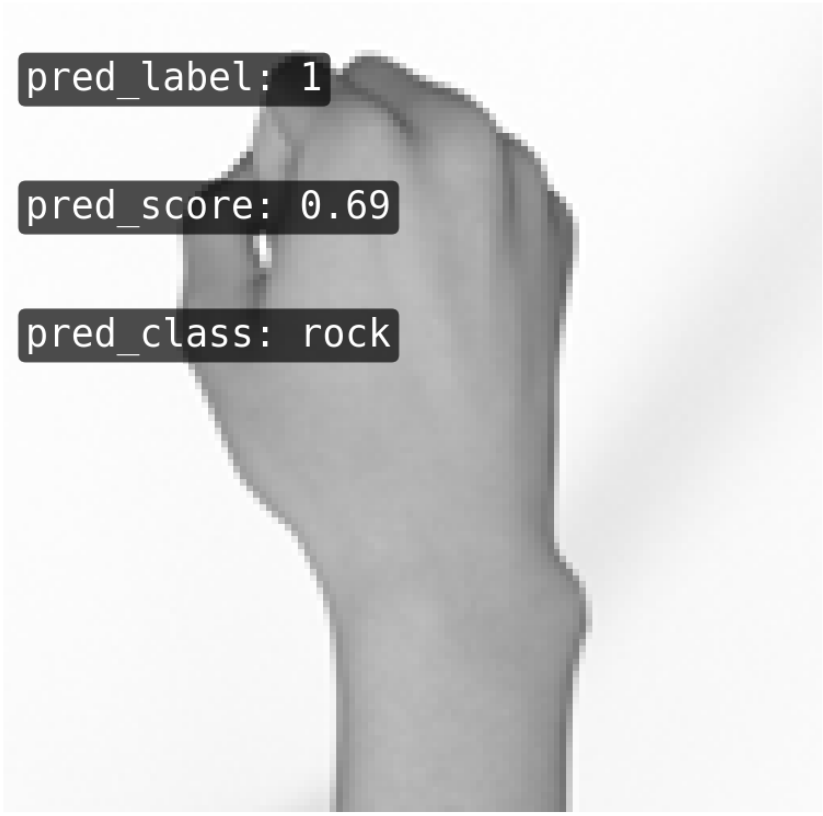
12.2.1 1. 模型推理

如果想快速上手体验 MMClassification 的话，我们建议您使用我们已经预训练好的模型和权重文件进行推理，提供一张图片测试推理的准确度。

执行代码如下:

```
import base # 测试版需要（发布版不需要）
from MMEdu import MMClassification as cls  # 导入mmcls模块

img = 'testrock01-02.png' # 指定进行推理的图片路径，我们使用demo文件夹中提供的图片
model = cls(backbone='LeNet') # 实例化MMClassification模型
model.checkpoint='../checkpoints/cls_model/hand_gray/latest.pth' #␣
↪指定使用的模型权重文件
class_path = '../dataset/classes/cls_classes.txt' # 指定训练集的路径
result = model.inference(image=img, show=True, class_path=class_path,␣
↪checkpoint=checkpoint) # 在CPU上进行推理
model.print_result() # 输出结果
# 同时您可以修改show的值来决定是否需要显示结果图片，此处默认显示结果图片
```



运行结果如图：

推理结果图片（带标签的图片）会以原来的文件名称保存在 demo 文件夹下的 cls_result 文件夹下，如果在 demo 下没有发现该文件夹，不用担心，系统会自动建立。当然，您可以自己指定保存文件夹的名称。

您也可以将收集的图片放在一个文件夹下，然后指定文件夹路径进行一组图片的批量推理。如在 demo 文件夹下新建一个 cls_testIMG 文件夹放图片，运行下面这段代码。

```
img = 'cls_testIMG/' # 指定进行推理的一组图片的路径
model = cls(backbone='LeNet') # 实例化MMClassification模型
model.checkpoint='../checkpoints/cls_model/hand_gray/latest.pth' #␣
↪指定使用的模型权重文件
class_path = '../dataset/classes/cls_classes.txt' # 指定训练集的路径
result = model.inference(image=img, show=True, class_path=class_path,␣
↪checkpoint=checkpoint) # 在CPU上进行推理
model.print_result() # 输出结果
# 同时您可以修改show的值来决定是否需要显示结果图片，此处默认显示结果图片
```

您会发现当前目录下‘cls_result’文件夹里出现了这组图片的推理结果图，每张图片的结果与您收集的图片同名，到这个文件夹下查看推理结果。

接下来对为您讲述代码规则：

- **图片准备**

```
img = 'testrock01-02.png' # 指定推理图片的路径，直接在代码所在的demo文件夹中选择图片
```

如果使用自己的图片的话，只需要修改 img 的路径即可（绝对路径和相对路径均可）

- **实例化模型**

```
model = cls(backbone='LeNet') # 实例化MMClassification模型
```

这里对于 MMClassification 模型提供的参数进行解释，MMClassification 支持传入的参数是 backbone。

backbone：指定使用的 MMClassification 模型，默认参数是'LeNet'，当然读者可以自行修改该参数以使用不同模型。

- **模型推理**

```
model.inference(image=img, show=True, class_path=class_path, checkpoint=checkpoint) #␣
↪在cpu上进行推理
```

将所需要推理图片的路径传入 inference 函数中即可进行推理，我们这里传入了四个参数，image 代表的就是推理图片的路径，show 代表是否需要显示结果图片，class_path 代表训练集的路径，checkpoint 代表指定使用的模型权重文件。

- **参数详解**

在 MMClassification 中对于 inference 函数还有其他的传入参数，在这里进行说明：

device：推理所用的设备，默认为'cpu'，如果电脑支持 GPU，也可以将参数修改为'cuda:0'，使用 GPU 进行推理。

checkpoint：指定使用的模型权重文件，默认参数为 None，如果没有指定模型权重文件，那么我们将会使用默认的模型权重文件进行推理。

image：推理图片的路径。

show：布尔值，默认为 True，表示推理后是否显示推理结果

class_path：指定训练集的路径，默认参数为'../dataset/classes/cls_classes.txt'。

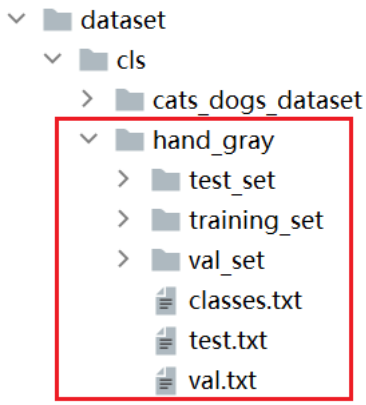
save_fold：保存的图片名，数据结构为字符串，默认参数为'cls_result'，用户也可以定义为自己想要的名字。

12.2.2 2. 训练模型

使用下面的代码即可简单体验 MMClassification 的训练过程，我们会为您进行详细的介绍。

在运行代码之前，您需要先拥有一个数据集，这里我们为您提供经典的石头剪刀布分类数据集。

数据集文件结构如下：



hand_gray 文件夹中包含三个图片文件夹，test_set,training_set,val_set 分别存储测试集，训练集和验证集的图片；以及三个 txt 文件，classes.txt 记录该数据集的类别，test.txt 和 val.txt 分别记录测试集和验证集的图片名。

- 代码展示

```
model = cls(backbone='LeNet') # 实例化模型，不指定参数即使用默认参数。
model.num_classes = 3 # 指定数据集中的类别数量
model.load_dataset(path='../dataset/cls/hand_gray') # 从指定数据集路径中加载数据
model.save_fold = '../checkpoints/cls_model/hand_gray' # 设置模型的保存路径
model.train(epochs=10, validate=True) # 设定训练的epoch次数以及是否进行评估
```

详细说明

实例化模型的代码在前面说过就不再赘述。

- **指定类别数量**

```
model.num_classes = 3 # 指定数据集中的类别数量
```

- **加载数据集**

```
model.load_dataset(path='../dataset/cls/hand_gray') # 从指定数据集路径中加载数据
```

这个函数只需要传入一个 path 参数即训练数据集的路径，函数的作用是修改模型中关于数据集路径的配置文件，从而确保我们在训练时不会找错文件。

- 指定模型参数存储位置

```
model.save_fold = '../checkpoints/cls_model/hand_gray'
```

- **模型训练**

```
model.train(epochs=10, validate=True) # 设定训练的epoch次数以及是否进行评估
```

表示训练 10 个轮次，并在训练结束后用校验集进行评估。

- **参数详解**

train 函数支持很多参数，为了降低难度，MMEdu 已经给绝大多数的参数设置了默认值。根据具体的情况修改参数，可能会得到更好的训练效果。下面来详细说明 train 函数的各个参数。

epochs：默认参数为 100，用于指定训练的轮次，而在上述代码中我们设置为 10。

validate：布尔值，只能为 True 或者 False，默认参数为 True，在训练结束后，设定是否需要在校验集上进行评估，True 则是需要进行评估。

random_seed：随机种子策略，默认为 0 即不使用，使用随机种子策略会减小模型算法结果的随机性。

save_fold：模型的保存路径，参数为 None，默认保存路径为./checkpoints/cls_model/，如果不想模型保存在该目录下，可自己指定路径。

distributed：布尔值，表示是否在分布式环境中训练该模型，默认为 False。

device：训练时所使用的设备，默认为'cpu'，如果电脑支持 GPU，也可以将参数修改为'cuda:0'，使用 GPU 进行推理。

metric：验证指标，默认参数为'accuracy'，在进行模型评估时会计算分类准确率，数值越高说明模型性能越好，我们在运行完程序之后也会看到这个结果。

save_best：验证指标，默认参数为'auto'，在进行模型评估时会计算分类准确率，数值越高说明模型性能越好，运行完程序之后会将这个结果保存。

optimizer：进行迭代时的优化器，默认参数为 SGD，SGD 会在训练的过程中迭代计算 mini-bath 的梯度。

lr：学习率，默认参数为 1e-2 即 0.01，指定模型进行梯度下降时的步长。简单解释就是，学习率过小，训练过程会很缓慢，学习率过大时，模型精度会降低。

checkpoint：指定使用的模型权重文件，默认参数为 None，如果没有指定模型权重文件，那么我们将会使用默认的模型权重文件进行推理。

执行上述代码之后的运行结果如下图：

```
2022-05-17 14:33:55,542 - mmcls - INFO - Checkpoints will be saved to G:\Project\checkpoints\cls_model\hand_gray by HardDiskBackend.
No CUDA runtime is found, using CUDA_HOME='C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v11.1'
No CUDA runtime is found, using CUDA_HOME='C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v11.1'
2022-05-17 14:34:09,161 - mmcls - INFO - Epoch [1][10/40]       lr: 1.000e-02, eta: 0:02:29, time: 1.362, data_time: 1.352, loss: 1.0977
2022-05-17 14:34:10,096 - mmcls - INFO - Epoch [1][20/40]       lr: 1.000e-02, eta: 0:01:12, time: 0.094, data_time: 0.085, loss: 1.0951
2022-05-17 14:34:11,045 - mmcls - INFO - Epoch [1][30/40]       lr: 1.000e-02, eta: 0:00:46, time: 0.095, data_time: 0.086, loss: 1.0894
2022-05-17 14:34:11,999 - mmcls - INFO - Epoch [1][40/40]       lr: 1.000e-02, eta: 0:00:32, time: 0.095, data_time: 0.087, loss: 1.0883
2022-05-17 14:34:12,252 - mmcls - INFO - Saving checkpoint at 1 epochs
[                                                  ] 0/33, elapsed: 0s, ETA:No CUDA runtime is found, using CUDA_HOME='C:\Program Files\NVIDIA GPU
No CUDA runtime is found, using CUDA_HOME='C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v11.1'
[>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>] 33/33, 5.9 task/s, elapsed: 6s, ETA:     0s2022-05-17 14:34:17,881 - mmcls - INFO - Epoch(val
ccuracy_top-1: 42.4242
No CUDA runtime is found, using CUDA_HOME='C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v11.1'
No CUDA runtime is found, using CUDA_HOME='C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v11.1'
2022-05-17 14:34:30,810 - mmcls - INFO - Epoch [2][10/40]       lr: 1.000e-02, eta: 0:00:41, time: 1.292, data_time: 1.283, loss: 1.0585
2022-05-17 14:34:30,937 - mmcls - INFO - Epoch [2][20/40]       lr: 1.000e-02, eta: 0:00:29, time: 0.013, data_time: 0.004, loss: 1.0593
2022-05-17 14:34:31,059 - mmcls - INFO - Epoch [2][30/40]       lr: 1.000e-02, eta: 0:00:21, time: 0.012, data_time: 0.004, loss: 1.0380
2022-05-17 14:34:31,172 - mmcls - INFO - Epoch [2][40/40]       lr: 1.000e-02, eta: 0:00:14, time: 0.011, data_time: 0.003, loss: 1.0126
2022-05-17 14:34:31,425 - mmcls - INFO - Saving checkpoint at 2 epochs
[>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>] 33/33, 1877.4 task/s, elapsed: 0s, ETA:     0s2022-05-17 14:34:31,452 - mmcls - INFO - Epoch(
ccuracy_top-1: 42.4242
No CUDA runtime is found, using CUDA_HOME='C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v11.1'
No CUDA runtime is found, using CUDA_HOME='C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v11.1'
2022-05-17 14:34:44,300 - mmcls - INFO - Epoch [3][10/40]       lr: 1.000e-02, eta: 0:00:14, time: 1.284, data_time: 1.276, loss: 0.9490
2022-05-17 14:34:44,425 - mmcls - INFO - Epoch [3][20/40]       lr: 1.000e-02, eta: 0:00:08, time: 0.012, data_time: 0.004, loss: 0.9164
2022-05-17 14:34:44,545 - mmcls - INFO - Epoch [3][30/40]       lr: 1.000e-02, eta: 0:00:03, time: 0.012, data_time: 0.003, loss: 0.8496
2022-05-17 14:34:44,655 - mmcls - INFO - Epoch [3][40/40]       lr: 1.000e-02, eta: 0:00:00, time: 0.011, data_time: 0.003, loss: 0.7503
```

而在 checkpoints\cls_model 文件夹中我们会发现多了两种文件，一个是 None.log.json 文件，它记录了我们模型在训练过程中的一些参数，比如说学习率 lr，所用时间 time，以及损失 loss 等；另一个文件是.pth 文件，这个是我们在训练过程中所保存的模型。

12.2.3 3. 继续训练

在这一步中，我们会教您加载之前训练过的模型接着训练，如果您觉得之前训练的模型 epoch 数不够的话或者因为一些客观原因而不得不提前结束训练，相信下面的代码会帮到您。

```
model = cls(backbone='LeNet') # 初始化实例模型
model.num_classes = 3 # 指定数据集中的类别数量
# model = cls(backbone='LeNet', num_classes = 3)
model.load_dataset(path='../dataset/cls/hand_gray') # 配置数据集路径
model.save_fold = '../checkpoints/cls_model/hand_gray' # 设置模型的保存路径
checkpoint = '../checkpoints/cls_model/hand_gray/latest.pth' # 指定使用的模型权重文件
model.train(epochs=50, validate=True, checkpoint=checkpoint) # 进行再训练
```

这里我们有一个参数在之前的训练模型过程中没有详细说明，那就是 train 函数中的 checkpoint 参数，这个放到这里就比较好理解，它的意思是指定需要进行再训练的模型路径，当然你也可以根据你需要训练的不同模型而调整参数。

12.2.4 4.SOTA 模型介绍

目前 MMClassifiation 支持的 SOTA 模型有 LeNet、MobileNet 等，这些模型的作用和适用场景简介如下。

- **LeNet**

适用于灰度图像识别。

- **MobileNet**

适用于绝大多数的图像识别，支持 1000 个分类。

- **ResNet**

广泛应用于分类、分割、检测等问题，结构简单，效果拔群。

CHAPTER 13

物体检测模块：MMDetection

13.1 简介

MMDetection 的主要功能：输出图片或视频中出现的多个对象名称，同时用方框框出对象所在方形区域。

其支持的 SOTA 模型有 FasterRCNN、Yolov3 等。

13.2 使用说明

MMEdu 中预置了“车牌检测”数据集，并且已经预训练了权重（路径：/checkpoint/det_model/plate/latest.pth）。在 demo 文件夹中，还提供了一张汽车图片（含车牌）用于测试。

13.2.1 1. 直接推理（支持 CPU）

如果想快速上手体验 MMDetection 的话，我们建议您使用我们已经预训练好的模型和权重文件进行推理，提供一张图片测试推理的准确度。

执行代码如下:

```
import base # 测试版需要（发布版不需要）
from MMEdu import MMDetection as det # 导入mmdet模块

img = 'car_plate.png' # 指定进行推理的图片路径，我们使用demo文件夹中提供的图片
model = det(backbone="FasterRCNN") # 实例化MMDetection模型
checkpoint = '../checkpoints/det_model/plate/latest.pth' # 指定使用的模型权重文件
class_path = '../dataset/classes/det_classes.txt' # 指定训练集的路径
result = model.inference(image=img, show=True, class_path=class_path,checkpoint_path
↪= checkpoint) # 在CPU上进行推理
model.print_result() # 输出结果
# 同时您可以修改show的值来决定是否需要显示结果图片，此处默认显示结果图片
```

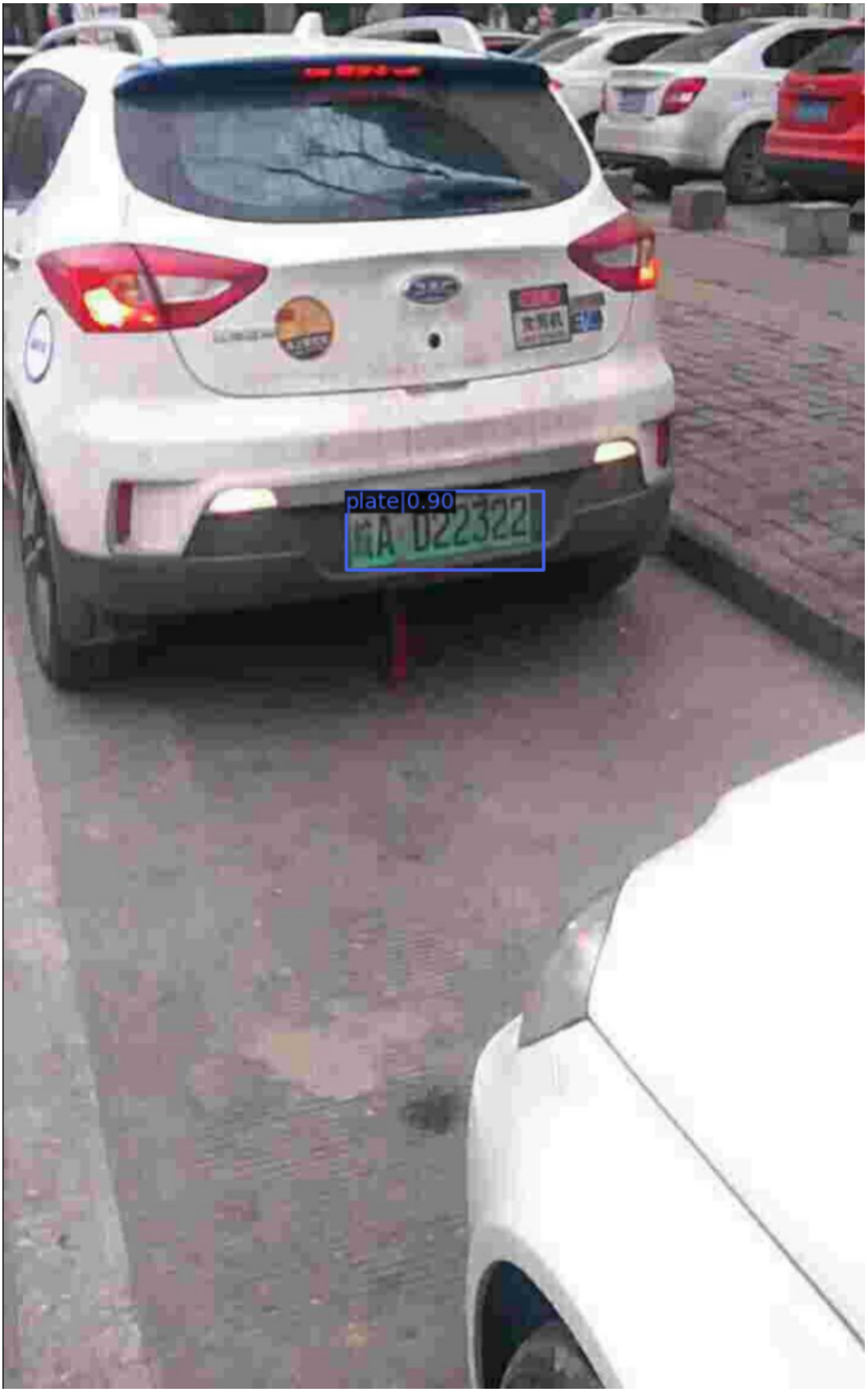


plate 0.90

运行结果如图：

推理结果图片（带标签的图片）会以原本的文件名称保存在 demo 文件夹下的 det_result 文件夹下，如果在 demo 下没有发现该文件夹，不用担心，系统会自动建立。当然，您可以自己指定保存文件夹的名称。

您也可以将收集的图片放在一个文件夹下，然后指定文件夹路径进行一组图片的批量推理。如在 demo 文件夹下新建一个 det_testIMG 文件夹放图片，运行下面这段代码。

```
img = 'det_testIMG/' # 指定进行推理的一组图片的路径
model = det(backbone="FasterRCNN") # 实例化MMDetection模型
checkpoint = '../checkpoints/det_model/plate/latest.pth' # 指定使用的模型权重文件
class_path = '../dataset/classes/det_classes.txt' # 指定训练集的路径
result = model.inference(image=img, show=True, class_path=class_path,checkpoint_path
↪= checkpoint) # 在CPU上进行推理
model.print_result() # 输出结果
# 同时您可以修改show的值来决定是否需要显示结果图片，此处默认显示结果图片
```

您会发现当前目录下‘det_result’文件夹里出现了这组图片的推理结果图，每张图片的结果与您收集的图片同名，您可以到这个文件夹下查看推理结果。

接下来对为您讲述代码规则：

- **图片准备**

```
img = 'car_plate.png' # 指定推理图片的路径，直接在代码所在的demo文件夹中选择图片
```

如果使用自己的图片的话，只需要修改 img 的路径即可（绝对路径和相对路径均可）。

- **实例化模型**

```
model = det(backbone='FasterRCNN') # 实例化MMDetection模型
```

这里对于 MMDetection 模型提供的参数进行解释，MMDetection 支持传入的参数是 backbone。

backbone：指定使用的 MMDetection 模型，默认使用 'FasterRCNN'，当然读者可以自行修改该参数以使用不同模型。

- **模型推理**

```
model.inference(image=img, show=True, class_path=class_path,checkpoint_path =␣
↪checkpoint) # 在CPU上进行推理
```

将所需要推理图片的路径传入 inference 函数中即可进行推理，我们这里传入了四个参数，image 代表的就是推理图片的路径，show 代表是否需要显示结果图片，class_path 代表训练集的路径，checkpoint 代表指定使用的模型权重文件。

- **参数详解**：

在 Detection_Edu 中对于 inference 函数还有其他的传入参数，在这里进行说明：

device：推理所用的设备，默认为'cpu'，如果电脑支持 GPU，也可以将参数修改为'cuda:0'，使用 GPU 进行推理。

checkpoint：指定使用的模型权重文件，默认参数为 None，如果没有指定模型权重文件，那么我们将会使用默认的模型权重文件进行推理。

image：推理图片的路径。

show：布尔值，默认为 True，表示推理后是否显示推理结果。

rpn_threshold & rcnn_threshold: 0 ~ 1 之间的数值。由于 FasterRCNN 为一个两阶段的检测模型，这两个参数分别表示两个阶段对于检测框的保留程度，高于这个数值的框将会被保留（这里如果同学们设置过低，也可能会发现图中出现了多个框）。

class_path：指定训练集的路径，默认参数为"../dataset/classes/det_classes.txt"。

save_fold：保存的图片名，数据结构为字符串，默认参数为'det_result'，用户也可以定义为自己想要的名字。

（最后两个参数的使用，我们将在下一部分进行详细举例解释）。

13.2.2 2. 训练模型

使用下面的代码即可简单体验 MMDetection 的训练过程，我们以车牌的识别为例，为您进行详细的介绍。

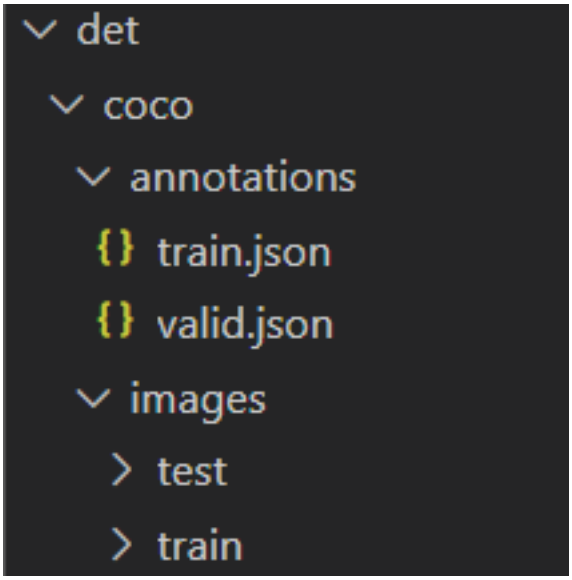
在运行代码之前，您需要先拥有一个数据集，这里我们为您提供车牌检测数据集。

数据集文件结构如下：

coco 文件夹中包含两个文件夹 annotations 和 images，分别存储标注信息以及图片数据，每个文件夹下面有 train 和 valid 两个 json 文件。

- 代码展示

```
model = det(backbone='FasterRCNN') # 实例化模型，不指定参数即使用默认参数。
model.num_classes = 1 # 进行车牌识别，此时只有一个类别。
model.load_dataset(path='../dataset/det/coco') # 从指定数据集路径中加载数据
model.save_fold = '../checkpoints/det_model/plate' # 设置模型的保存路径
model.train(epochs=3, validate=True) # 设定训练的epoch次数以及是否进行评估
```



详细说明

实例化模型的代码在前面说过就不再赘述。

- **指定类别数量**

```
model.num_classes = 1 # 进行车牌识别，此时只有一个类别
```

- **加载数据集**

```
model.load_dataset(path='../dataset/det/coco') # 从指定数据集路径中加载数据
```

这个函数只需要传入一个 path 参数即训练数据集的路径，函数的作用是修改模型中关于数据集路径的配置文件，从而确保我们在训练时不会找错文件。

- **指定模型参数存储位置**

```
model.save_fold = '../checkpoints/det_model/plate'
```

- **模型训练**

```
model.train(epochs=10, validate=True) # 设定训练的epoch次数以及是否进行评估
```

表示训练 10 个轮次，并在训练结束后用检验集进行评估。

- **参数详解**

train 函数支持很多参数，为了降低难度，MMEdu 已经给绝大多数的参数设置了默认值。根据具体的情况修改参数，可能会得到更好的训练效果。下面来详细说明 train 函数的各个参数。

random_seed：随机种子策略，默认为 0 即不使用，使用随机种子策略会减小模型算法结果的随机性。

save_fold：模型的保存路径，默认参数为./checkpoints/det_model/，如果不想模型保存在该目录下，可自己指定路径。

distributed：布尔值，只能为 True 或者 False，默认参数为 False，设为 True 时即使用分布式训练。

epochs：默认参数为 100，用于指定训练的轮次，而在上述代码中我们设置为 10。

validate：布尔值，只能为 True 或者 False，默认参数为 True，在训练结束后，设定是否需要在校验集上进行评估，True 则是需要进行评估。

metric：验证指标，默认参数为'bbox'，在进行模型评估时会计算预测的检测框和实际检测框相交的多少，数值越高说明模型性能越好，我们在运行完程序之后也会看到这个结果。

save_best：验证指标，默认参数为'bbox_mAP'，在进行模型评估时会计算分类准确率，数值越高说明模型性能越好，运行完程序之后会将这个结果保存。

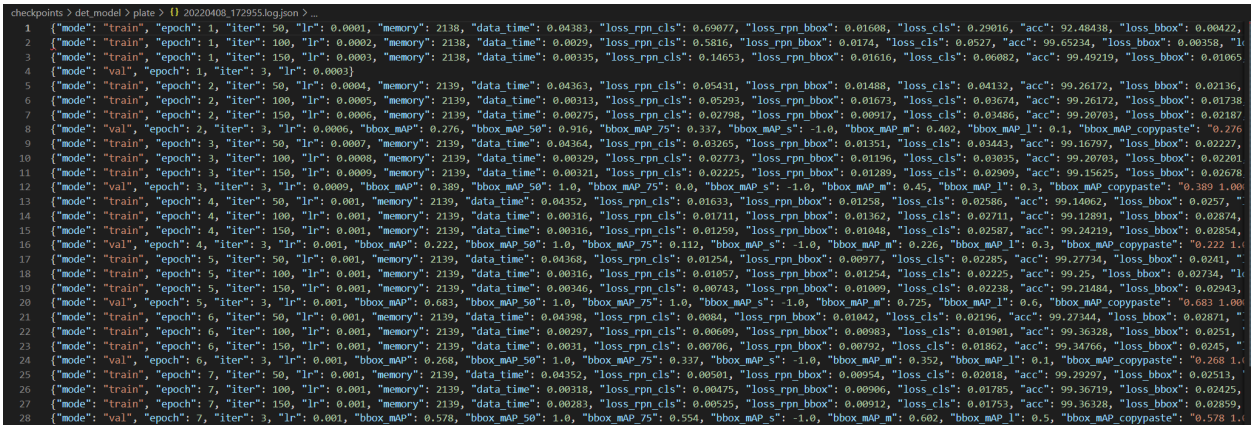
optimizer：进行迭代时的优化器，默认参数为 SGD，SGD 会在训练的过程中迭代计算 mini-bath 的梯度。

lr：学习率，默认参数为 1e-3 即 0.001，指定模型进行梯度下降时的步长。简单解释就是，学习率过小，训练过程会很缓慢，学习率过大时，模型精度会降低。

weight_decay：权值衰减参数，用来调节模型复杂度对损失函数的影响，防止过拟合，默认值为 1e-3 即 0.001。

checkpoint: 默认为’None’，表示在训练过程中使用初始化权重。如果使用训练得到的模型（或预训练模型），此参数传入一个模型路径，我们的训练将基于传入的模型参数继续训练。

执行上述代码之后的运行结果如下图



而在 checkpoints\det_model 文件夹中我们会发现多了两种文件，一个是 None.log.json 文件，它记录了我们模型在训练过程中的一些参数，比如说学习率 lr，所用时间 time，以及损失 loss 等；另一个文件是.pth 文件，这个是我们在训练过程中所保存的模型。

13.2.3 3. 继续训练

在这一步中，我们会教您加载之前训练过的模型接着训练，如果您觉得之前训练的模型 epoch 数不够的话或者因为一些客观原因而不得不提前结束训练，相信下面的代码会帮到您。

```
model = det(backbone='FasterRCNN') # 初始化实例模型
model.num_classes = 1  # 进行车牌识别，此时只有一个类别。
model.load_dataset(path='../dataset/det/coco') # 配置数据集路径
model.save_fold = '../checkpoints/det_model/plate' # 设置模型的保存路径
checkpoint='../checkpoints/det_model/plate/latest.pth' # 指定使用的模型权重文件
model.train(epochs=3, validate=True, checkpoint=checkpoint) # 进行再训练
```

这里我们有一个参数在之前的训练模型过程中没有提及，那就是 train 函数中的 checkpoint 参数，这个放到这里就比较好理解，它的意思是指定需要进行再训练的模型路径，当然你也可以根据你需要训练的不同模型而调整参数。

13.2.4 4.SOTA 模型介绍

目前 MMDetection 支持的 SOTA 模型有 FaterRCNN、Yolov3 等，这些模型的作用和适用场景简介如下。

- **FasterRCNN**

采用双阶检测方法，可以解决多尺度、小目标问题，通用性强。

- **Yolov3**

只进行一次检测，速度较快，适用于稍微大的目标检测问题。

CHAPTER 14

自定义网络模块：MMBase

14.1 0. 引入包

```
from MMEdu.MMBase import *
```

14.2 1. 声明模型

```
model = MMBase()
```

14.3 2. 载入数据

此处采用 lvis 鸢尾花数据集作为示例。

```
# 训练数据
train_path = '../dataset/iris/iris_training.csv'
x = np.loadtxt(train_path, dtype=float, delimiter=',',skiprows=1,usecols=range(0,4))
↪# 读取前四列，特征
y = np.loadtxt(train_path, dtype=int, delimiter=',',skiprows=1,usecols=4) #␣
↪读取第五列，标签
# 测试数据
test_path = '../dataset/iris/iris_test.csv'
test_x = np.loadtxt(test_path, dtype=float, delimiter=',',skiprows=1,usecols=range(0,
↪4)) # 读取前四列，特征
test_y = np.loadtxt(test_path, dtype=int, delimiter=',',skiprows=1,usecols=4) #␣
↪读取第五列，标签
# 将数据载入
model.load_dataset(x, y)
```

14.4 3. 搭建模型

逐层添加，搭建起模型结构。注释标明了数据经过各层的尺寸变化。

```
model.add(layer='Linear',size=(4, 10),activation='ReLU') # [120, 10]
model.add(layer='Linear',size=(10, 5), activation='ReLU') # [120, 5]
model.add(layer='Linear', size=(5, 3), activation='Softmax') # [120, 3]
```

以上使用 `add()` 方法添加层，参数 `layer='Linear'` 表示添加的层是线性层，`size=(4,10)` 表示该层输入维度为 4，输出维度为 10，`activation='ReLU'` 表示使用 ReLU 激活函数。更详细 `add()` 方法使用可见附录 1。

14.5 4. 模型训练

模型训练可以采用以下函数：

```
model.train(lr=0.01, epochs=500,checkpoint=checkpoint)
```

参数 lr 为学习率，epochs 为训练轮数，checkpoint 为现有模型路径，当使用 checkpoint 参数时，模型基于一个已有的模型继续训练，不使用 checkpoint 参数时，模型从零开始训练。

14.5.1 4.1 正常训练

```
model = MMBase()
model.add(layer='Linear',size=(4, 10),activation='ReLU') # [120, 10]
model.add(layer='Linear',size=(10, 5), activation='ReLU') # [120, 5]
model.add(layer='Linear', size=(5, 3), activation='Softmax') # [120, 3]
model.load_dataset(x, y)
model.save_fold = 'checkpoints'
model.train(lr=0.01, epochs=1000)
```

model.save_fold 表示训练出的模型文件保存的文件夹。

14.5.2 4.2 继续训练

```
model = MMBase()
model.load_dataset(x, y)
model.save_fold = 'checkpoints'
checkpoint = 'checkpoints/mmbase_net.pkl'
model.train(lr=0.01, epochs=1000, checkpoint=checkpoint)
```

14.6 5. 使用现有模型直接推理

可使用以下函数进行推理：

```
model.inference(data=test_x, checkpoint=checkpoint)
```

参数 data 为待推理的测试数据数据，该参数必须传入值；

checkpoint 为已有模型路径，即使用现有的模型进行推理，该参数可以不传入值，即直接使用训练出的模型做推理。

```
model = MMBase() # 声明模型
checkpoint = 'checkpoints/mmbase_net.pkl' # 现有模型路径
result = model.inference(data=test_x, checkpoint=checkpoint) # 直接推理
model.print_result() # 输出结果
```

14.7 6. 输出推理结果

```
res = model.inference(test_x)
```

输出结果数据类型为 numpy 的二维数组，表示各个样本的各个特征的置信度。

```
model.print_result() # 输出字典格式结果
```

输出结果数据类型为字典，格式为 {样本编号：{预测值：x，置信度：y}}。该函数调用即输出，但也有返回值。

14.8 7. 模型的保存与加载

```
# 保存
model.save("mmbase_net.pkl")
# 加载
model.load("mmbase_net.pkl")
```

参数为模型保存的路径，`.pkl` 文件格式可以理解为将 python 中的数组、列表等持久化地存储在硬盘上的一种方式。

注：train()，inference() 函数中也可通过参数控制模型的保存与加载，但这里也列出单独保存与加载模型的方法，以确保灵活性。

14.9 8. 查看模型结构

```
model.print_model()
```

无参数。

完整测试用例可见 mmbase_demo.py 文件。

14.10 附录

14.10.1 1. add() 详细介绍

此处以典型的 LeNet5 网络结构为例。注释标明了数据经过各层的尺寸变化。

```
model.add('Conv2D', size=(1, 3),kernel_size=( 3, 3)) # [100, 3, 18, 18]
model.add('MaxPool', kernel_size=(2,2), activation='ReLU') # [100, 3, 9, 9]
model.add('Conv2D', size=(3, 10), kernel_size=(3, 3)) # [100, 10, 7, 7]
model.add('AvgPool', kernel_size=(2,2), activation='ReLU') # [100, 10, 3, 3]
model.add('Linear', size=(90, 10), activation='ReLU') # [100, 10]
model.add('Linear', size=(10, 2), activation='Softmax') # [100,2]
model.add(optimizer='SGD') # 设定优化器
```

添加层的方法为 `add(self, layer=None, activation=None, optimizer='SGD', **kw)`,

参数:

layer：层的类型，可选值包括 Conv2D, MaxPool, AvgPool, Linear。

activation：激活函数类型，可选值包括 ReLU，Softmax。

optimizer：为优化器类型，默认值为 SGD，可选值包括 SGD，Adam，Adagrad，ASGD。

kw：关键字参数，包括与 size 相关的各种参数，常用的如 size=(x,y)，x 为输入维度，y 为输出维度；kernel_size=(a,b)，(a,b) 表示核的尺寸。

以下具体讲述各种层：

Conv2D：卷积层（二维），需给定 size，kernel_size。

MaxPool：最大池化层，需给定 kernel_size。

AvgPool：平均池化层，需给定 kernel_size。

Linear：线性层，需给定 size。

CHAPTER 15

MMEdu 的数据集格式详解

MMEdu 系列提供了包括分类、检测等任务的若干数据集，存储在 dataset 文件夹下。

15.1 1.ImageNet

ImageNet 是斯坦福大学提出的一个用于视觉对象识别软件研究的大型可视化数据库，目前大部分模型的性能基准测试都在 ImageNet 上完成。MMEdu 的 MMClassification 支持的数据集类型是 ImageNet，如需训练自己创建的数据集，数据集需转换成 ImageNet 格式。

ImageNet 格式数据集文件夹结构如下所示，图像数据文件夹和标签文件放在同级目录下。

```
imagenet
├── ...
├── training_set
|   ├── class_0
|   |   ├── filesname_0.JPEG
|   |   ├── filesname_1.JPEG
|   |   ├── ...
|   ├── ...
|   ├── class_n
|   |   ├── filesname_0.JPEG
|   |   ├── filesname_1.JPEG
|   |   ├── ...
├── classes.txt
├── ...
```

如上所示训练数据根据图片的类别，存放至不同子目录下，子目录名称为类别名称。

classes.txt 包含数据集类别标签信息，每行包含一个类别名称，按照字母顺序排列。

```
class_0
class_1
...
class_n
```

为了验证和测试，我们建议划分训练集、验证集和测试集，此时需另外生成“val.txt”和“test.txt”这两个标签文件，要求是每一行都包含一个文件名和其相应的真实标签。格式如下所示：

```
filesname_0.jpg 0
filesname_1.jpg 0
...
filesname_a.jpg n
filesname_b.jpg n
```

注：真实标签的值应该位于 [0, 类别数目-1] 之间。

这里，为您提供一段用 Python 代码完成标签文件的程序如下所示，程序中设计了“val.txt”和“test.txt”这两个标签文件每行会包含类别名称、文件名和真实标签。

```
import os
# 列出指定目录下的所有文件名，确定类别名称
classes = os.listdir('D:\测试数据集\EX_dataset\\training_set')
# 打开指定文件，并写入类别名称
with open('D:\测试数据集\EX_dataset/classes.txt','w') as f:
    for line in classes:
        str_line = line +'\n'
        f.write(str_line) # 文件写入str_line，即类别名称

test_dir = 'D:\测试数据集\EX_dataset\\test_set/' # 指定测试集文件路径
# 打开指定文件，写入标签信息
with open('D:\测试数据集\EX_dataset/test.txt','w') as f:
    for cnt in range(len(classes)):
        t_dir = test_dir + classes[cnt]  # 指定测试集某个分类的文件目录
        files = os.listdir(t_dir) # 列出当前类别的文件目录下的所有文件名
        # print(files)
        for line in files:
            str_line = classes[cnt] + '/' + line + ' '+str(cnt) +'\n'
            f.write(str_line)

val_dir = 'D:\测试数据集\EX_dataset\\val_set/'  # 指定文件路径
# 打开指定文件，写入标签信息
with open('D:\测试数据集\EX_dataset/val.txt', 'w') as f:
    for cnt in range(len(classes)):
        t_dir = val_dir + classes[cnt]  # 指定验证集某个分类的文件目录
        files = os.listdir(t_dir)  # 列出当前类别的文件目录下的所有文件名
        # print(files)
        for line in files:
            str_line = classes[cnt] + '/' + line + ' ' + str(cnt) + '\n'
            f.write(str_line)  # 文件写入str_line，即标注信息
```

至于如何从零开始制作一个 ImageNet 格式的数据集，可参考如下步骤。

15.1.1 第一步：整理图片

您可以用任何设备拍摄图像，也可以从视频中抽取帧图像，需要注意，这些图像可以被划分为多个类别。每个类别建立一个文件夹，文件夹名称为类别名称，将图片放在其中。

接下来需要对图片进行尺寸、保存格式等的统一，可使用如下代码：

```
from PIL import Image
from torchvision import transforms
import os
```

(续下页)

(接上页)

```
def makeDir(folder_path):
    if not os.path.exists(folder_path):  # 判断是否存在文件夹如果不存在则创建为文件夹
        os.makedirs(folder_path)

classes = os.listdir('D:\测试数据集\自定义数据集')
read_dir = 'D:\测试数据集\自定义数据集/' # 指定原始图片路径
new_dir = 'D:\测试数据集\自定义数据集new/'
for cnt in range(len(classes)):
    r_dir = read_dir + classes[cnt] + '/'
    files = os.listdir(r_dir)
    for index,file in enumerate(files):
        img_path = r_dir + file
        img = Image.open(img_path)   # 读取图片
        resize = transforms.Resize([224, 224])
        IMG = resize(img)
        w_dir = new_dir + classes[cnt] + '/'
        makeDir(w_dir)
        save_path = w_dir + str(index)+'.jpg'
        IMG = IMG.convert('RGB')
        IMG.save(save_path)
```

15.1.2 第二步：划分训练集、验证集和测试集

根据整理的数据集大小，按照一定比例拆分训练集、验证集和测试集，可使用如下代码将原始数据集按照“6:2:2”的比例拆分。

```
import os
import shutil
# 列出指定目录下的所有文件名，确定类别名称
classes = os.listdir('D:\测试数据集\自定义表情数据集')

# 定义创建目录的方法
def makeDir(folder_path):
    if not os.path.exists(folder_path):  # 判断是否存在文件夹如果不存在则创建为文件夹
        os.makedirs(folder_path)

# 指定文件目录
read_dir = 'D:\测试数据集\自定义表情数据集/' # 指定原始图片路径
train_dir = 'D:\测试数据集\自制\EX_dataset\\training_set/' # 指定训练集路径
test_dir = 'D:\测试数据集\自制\EX_dataset\\test_set/' # 指定测试集路径
val_dir = 'D:\测试数据集\自制\EX_dataset\\val_set/' # 指定验证集路径

for cnt in range(len(classes)):
    r_dir = read_dir + classes[cnt] + '/'  # 指定原始数据某个分类的文件目录
    files = os.listdir(r_dir)  # 列出某个分类的文件目录下的所有文件名
    files = files[:1000]
    # 按照6:2:2拆分文件名
    offset1 = int(len(files) * 0.6)
    offset2 = int(len(files) * 0.8)
    training_data = files[:offset1]
    val_data = files[offset1:offset2]
    test_data = files[offset2:]

    # 根据拆分好的文件名新建文件目录放入图片
```

(续下页)

(接上页)

```
    for index,fileName in enumerate(training_data):
        w_dir = train_dir + classes[cnt] + '/'  # 指定训练集某个分类的文件目录
        makeDir(w_dir)
        shutil.copy(r_dir + fileName,w_dir + classes[cnt] + str(index)+'.jpg')
    for index,fileName in enumerate(test_data):
        w_dir = test_dir + classes[cnt] + '/'  # 指定测试集某个分类的文件目录
        makeDir(w_dir)
        shutil.copy(r_dir + fileName, w_dir + classes[cnt] + str(index) + '.jpg')
    for index,fileName in enumerate(val_data):
        w_dir = val_dir + classes[cnt] + '/'  # 指定验证集某个分类的文件目录
        makeDir(w_dir)
        shutil.copy(r_dir + fileName, w_dir + classes[cnt] + str(index) + '.jpg')
```

15.1.3 第三步：生成标签文件

划分完训练集、验证集和测试集，我们需要生成“classes.txt”，“val.txt”和“test.txt”，使用上文介绍的 Python 代码完成标签文件的程序生成标签文件。

15.1.4 第四步：给数据集命名

最后，我们将这些文件放在一个文件夹中，命名为数据集的名称。这样，在训练的时候，只要通过 `model.load_dataset` 指定数据集的路径就可以了。

15.2 2.COCO

COCO 数据集是微软于 2014 年提出的一个大型的、丰富的检测、分割和字幕数据集，包含 33 万张图像，针对目标检测和实例分割提供了 80 个类别的物体的标注，一共标注了 150 万个物体。MMEdu 的 MMDetection 支持的数据集类型是 COCO，如需训练自己创建的数据集，数据集需转换成 COCO 格式。

MMEdu 的 MMDetection 设计的 COCO 格式数据集文件夹结构如下所示，“annotations”文件夹存储标注文件，“images”文件夹存储用于训练、验证、测试的图片。

```
coco
├── annotations
│   ├── train.json
│   ├── ...
├── images
│   ├── train
│   │   ├── filesname_0.JPEG
│   │   ├── filesname_1.JPEG
│   │   ├── ...
│   ├── ...
```

如果您的文件夹结构和上方不同，则需要在“Detection_Edu.py”文件中修改 `load_dataset` 方法中的数据集和标签加载路径。

COCO 数据集的标注信息存储在“annotations”文件夹中的 `json` 文件中，需满足 COCO 标注格式，基本数据结构如下所示。

```
# 全局信息
{
    "images": [image],
    "annotations": [annotation],
    "categories": [category]
}

# 图像信息标注，每个图像一个字典
image {
    "id": int,  # 图像id编号，可从0开始
    "width": int, # 图像的宽
    "height": int,  # 图像的高
    "file_name": str, # 文件名
}

# 检测框标注，图像中所有物体及边界框的标注，每个物体一个字典
annotation {
    "id": int,  # 注释id编号
    "image_id": int,  # 图像id编号
    "category_id": int,   # 类别id编号
    "segmentation": RLE or [polygon],  # 分割具体数据，用于实例分割
    "area": float,  # 目标检测的区域大小
    "bbox": [x,y,width,height],  # 目标检测框的坐标详细位置信息
    "iscrowd": 0 or 1,  # 目标是否被遮盖，默认为0
}

# 类别标注
categories [{
    "id": int, # 类别id编号
    "name": str, # 类别名称
    "supercategory": str, # 类别所属的大类，如哈巴狗和狐狸犬都属于犬科这个大类
}]
```

这里，为您提供一种自己制作 COCO 格式数据集的方法。

15.2.1 第一步、整理图片

根据需求按照自己喜欢的方式收集图片，图片中包含需要检测的信息即可，可以使用 ImageNet 格式数据集整理图片的方式对收集的图片进行预处理。

15.2.2 第二步、标注图片

可使用 LabelMe 批量打开图片文件夹的图片，进行标注并保存为 json 文件。

- LabelMe：格式为 LabelMe，提供了转 VOC、COCO 格式的脚本，可以标注矩形、圆形、线段、点。标注语义分割、实例分割数据集尤其推荐。
- LabelMe 安装与打开方式：`pip install labelme` 安装完成后输入 `labelme` 即可打开。

15.2.3 第三步、转换成 COCO 标注格式

将 LabelMe 格式的标注文件转换成 COCO 标注格式，可以使用如下代码：

```
import json
import numpy as np
import glob
import PIL.Image
from PIL import ImageDraw
from shapely.geometry import Polygon

class labelme2coco(object):
    def __init__(self, labelme_json=[], save_json_path='./new.json'):
        '''
        :param labelme_json: 所有labelme的json文件路径组成的列表
        :param save_json_path: json保存位置
        '''
        self.labelme_json = labelme_json
        self.save_json_path = save_json_path
        self.annotations = []
        self.images = []
        self.categories = [{'supercategory': None, 'id': 1, 'name': 'cat'},{
↪'supercategory': None, 'id': 2, 'name': 'dog'}] # 指定标注的类别
        self.label = []
        self.annID = 1
        self.height = 0
        self.width = 0
        self.save_json()

    # 定义读取图像标注信息的方法
    def image(self, data, num):
        image = {}
        height = data['imageHeight']
        width = data['imageWidth']
        image['height'] = height
        image['width'] = width
        image['id'] = num + 1
        image['file_name'] = data['imagePath'].split('/')[-1]
        self.height = height
        self.width = width
        return image

    # 定义数据转换方法
    def data_transfer(self):
        for num, json_file in enumerate(self.labelme_json):
            with open(json_file, 'r') as fp:
                data = json.load(fp)  # 加载json文件
                self.images.append(self.image(data, num)) #␣
↪读取所有图像标注信息并加入images数组
                for shapes in data['shapes']:
                    label = shapes['label']
                    points = shapes['points']
                    shape_type = shapes['shape_type']
                    if shape_type == 'rectangle':
                        points = [points[0],[points[0][0],points[1][1]],points[1],
↪[points[1][0],points[0][1]]]
                    self.annotations.append(self.annotation(points, label, num)) #␣
↪读取所有检测框标注信息并加入annotations数组
```

(续下页)

(接上页)

```
                    self.annID += 1
        print(self.annotations)

    # 定义读取检测框标注信息的方法
    def annotation(self, points, label, num):
        annotation = {}
        annotation['segmentation'] = [list(np.asarray(points).flatten())]
        poly = Polygon(points)
        area_ = round(poly.area, 6)
        annotation['area'] = area_
        annotation['iscrowd'] = 0
        annotation['image_id'] = num + 1
        annotation['bbox'] = list(map(float, self.getbbox(points)))
        annotation['category_id'] = self.getcatid(label)
        annotation['id'] = self.annID
        return annotation

    # 定义读取检测框的类别信息的方法
    def getcatid(self, label):
        for categorie in self.categories:
            if label == categorie['name']:
                return categorie['id']
        return -1

    def getbbox(self, points):
        polygons = points
        mask = self.polygons_to_mask([self.height, self.width], polygons)
        return self.mask2box(mask)

    def mask2box(self, mask):
        '''从mask反算出其边框
        mask：[h,w]  0、1组成的图片

↪1对应对象，只需计算1对应的行列号（左上角行列号，右下角行列号，就可以算出其边框）
        '''
        # np.where(mask==1)
        index = np.argwhere(mask == 1)
        rows = index[:, 0]
        clos = index[:, 1]
        # 解析左上角行列号
        left_top_r = np.min(rows)  # y
        left_top_c = np.min(clos)  # x

        # 解析右下角行列号
        right_bottom_r = np.max(rows)
        right_bottom_c = np.max(clos)

        return [left_top_c, left_top_r, right_bottom_c - left_top_c,
                right_bottom_r - left_top_r]  # [x1,y1,w,h] 对应COCO的bbox格式

    def polygons_to_mask(self, img_shape, polygons):
        mask = np.zeros(img_shape, dtype=np.uint8)
        mask = PIL.Image.fromarray(mask)
        xy = list(map(tuple, polygons))
        PIL.ImageDraw.Draw(mask).polygon(xy=xy, outline=1, fill=1)
        mask = np.array(mask, dtype=bool)
```

(续下页)

(接上页)

```
        return mask

    def data2coco(self):
        data_coco = {}
        data_coco['images'] = self.images
        data_coco['categories'] = self.categories
        data_coco['annotations'] = self.annotations
        return data_coco

    def save_json(self):
        self.data_transfer()
        self.data_coco = self.data2coco()
        # 保存json文件
        json.dump(self.data_coco, open(self.save_json_path, 'w'), indent=4)  #␣
↪写入指定路径的json文件，indent=4 更加美观显示

labelme_json = glob.glob('picture/*.json')  # 获取指定目录下的json格式的文件
labelme2coco(labelme_json, 'picture/new.json') # 指定生成文件路径
```

15.2.4 第四步、按照目录结构整理文件

创建两个文件夹“images”和“annotations”，分别用于存放图片以及标注信息。按照要求的目录结构，整理好文件夹的文件，最后将文件夹重新命名，在训练的时候，只要通过 `model.load_dataset` 指定数据集的路径就可以了。

CHAPTER 16

数据集标注工具

稍后更新

CHAPTER 17

Web 库 PyWebIO 简介

17.1 1. 简介

顾名思义，PyWebIO 库是一个基于 Web 方式来实现输入输出（I/O）的 Python 库。这是北京航空航天大学在读研究生王伟民同学用业余时间写的库。目前在 GitHbu 上获得了高达 1.6K 的 Star。它允许用户像编写终端脚本一样来编写 Web 应用或基于浏览器的 GUI 应用，而无需具备 HTML 和 JS 的相关知识。

Github 地址：https://github.com/wang0618/PyWebIO

17.2 2. 安装

PyWebIO 可以采用 pip 命令安装，具体如下：

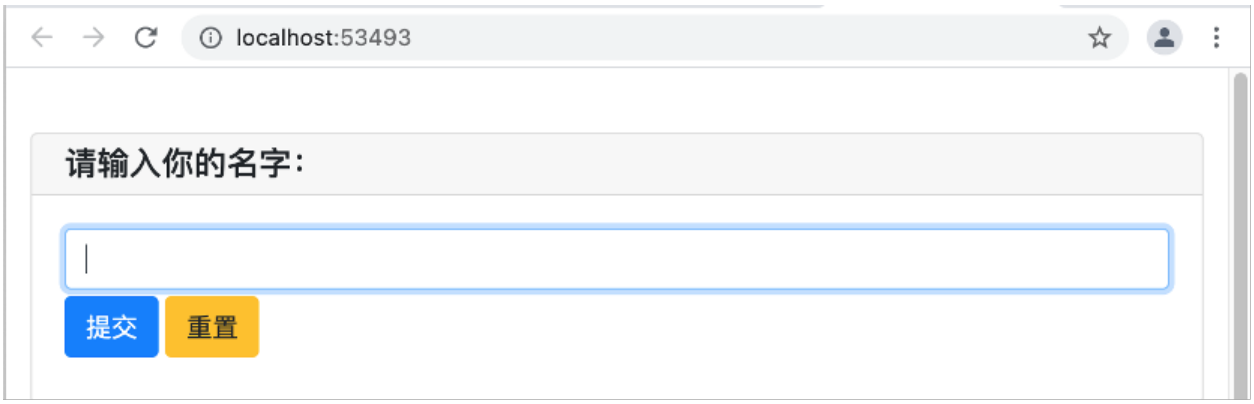
```
pip install PyWebIO
```

注：MMEdu 中已经内置了 PyWebIO 库。

17.3 3. 代码示例

PyWebIO 提供了一系列命令式的交互函数来在浏览器上获取用户输入和进行输出，相当于将浏览器变成了一个“富文本终端”。如：

```
from pywebio.input import *
from pywebio.output import *
# 文本输入
s = input('请输入你的名字：')
# 输出文本
put_text('欢迎你，' + s);
```



运行这段代码后，浏览器会自动打开一个本地的网址，出现这样的界面。

图 1 初始网页界面

输入姓名再点击“提交”按钮，网页上就会输出相应的文字。这种基于 Web 页面的“交互”，体验比黑乎乎的终端界面要好很多。

PyWebIO 支持常见的网页控件。既然 PyWebI 的定位就是输入和输出，那么也可以将网页控件分为这两类，部分控件的说明如表 1 所示。

表 1 PyWebIO 支持的网页控件（部分）

类别

控件

代码范例

输入

文本

input（“What’s your name?”）

下拉选择

select（‘Select’, [‘A’, ‘B’]）

多选

checkbox（“Checkbox”, options=[‘Check me’]）

单选

radio（“Radio”, options=[‘A’, ‘B’, ‘C’]）

多行文本

textarea（‘Text’, placeholder=‘Some text’）

文件上传

file_upload（“Select a file:”）

输出

文本

put_text（“Hello world!”）;

表格

put_table([[‘Product’, ‘Price’],[‘Apple’, ‘$5.5’], [‘Banner’, ‘$7’],]);

图像

put_image(open(‘python-logo.png’, ‘rb’).read());

通知消息

toast(‘Awesome PyWebIO!!’);

文件

put_file(‘hello_word.txt’, b’hello word!’);

Html 代码

put_html(‘E = mc2’);

尤其值得称赞的是，PyWebIO 还支持 MarkDown 语法。除了输入输出，PyWebIO 还支持布局、协程、数据可视化等特性。通过和其他库的配合，可以呈现更加酷炫的网页效果，如图 2 所示。

图 2 PyWebIO 结合第三方库制作的数据可视化效果

如果需要了解更多关于 PyWebIO 库的资源，请访问 github 或者官方文档。

文档地址：https://pywebio.readthedocs.io/

17.4 4. 借助 PyWebIO 部署简易 AI 应用

在人工智能教学过程中，我们常常为模型的部署而烦恼。如果训练出来的模型不能有效应用于生活，或者解决一些真实问题，则很难打动学生，激发学习兴趣。

PyWebIO 能够将 AI 模型快速“变身”为 Web 应用，上传一张照片就能输出识别结果，极大地提高了学生的学习收获感。

CHAPTER 18

MQTT 库 siot 简介

18.1 1. 简介

MQTT 是最常用的物联网协议之一。但是，MQTT 的官方 Python 库明显不好用，前面要定义一个类，代码冗长，对初学者不够友好。siot 是虚谷物联团队基于 MQTT paho 写的一个 Python 库，为了让初学者能够写出更加简洁、优雅的 Python 代码。

需要强调的是，siot 库同时支持 MicroPython，语法完全一致。

GitHub 地址：https://github.com/vvlink/SIoT/tree/master/siot-lib

18.2 2. 安装

可以使用使用 pip 命令安装 siot 库，如：

```
pip install siot
```

注：MMEdu 中已经内置了 siot 库。

18.3 3. 代码范例

下面的代码以 MQTT 服务器软件 SIoT 为例。SIoT 是一个一键部署的 MQTT 服务器，广泛应用于中小学的物联网教学中。

18.3.1 3.1 发送消息

```
import siot
import time

SERVER = "127.0.0.1"            #MQTT服务器IP
CLIENT_ID = ""                  #在SIoT上，CLIENT_ID可以留空
IOT_pubTopic  = 'xzr/001'       #“topic”为“项目名称/设备名称”
IOT_UserName ='siot'            #用户名
IOT_PassWord ='dfrobot'         #密码

siot.init(CLIENT_ID, SERVER, user=IOT_UserName, password=IOT_PassWord)
siot.connect()
siot.loop()

tick = 0
while True:
  siot.publish(IOT_pubTopic, "value %d"%tick)
  time.sleep(1)           #隔1秒发送一次
  tick = tick+1
```

18.3.2 3.2 订阅消息

```
import siot
import time

SERVER = "127.0.0.1"        #MQTT服务器IP
CLIENT_ID = ""              #在SIoT上，CLIENT_ID可以留空
IOT_pubTopic  = 'xzr/001'   #“topic”为“项目名称/设备名称”
IOT_UserName ='siot'        #用户名
IOT_PassWord ='dfrobot'     #密码

def sub_cb(client, userdata, msg):
  print("\nTopic:" + str(msg.topic) + " Message:" + str(msg.payload))
  # msg.payload中是消息的内容，类型是bytes，需要用解码。
  s=msg.payload.decode()
  print(s)

siot.init(CLIENT_ID, SERVER, user=IOT_UserName, password=IOT_PassWord)
siot.connect()
siot.subscribe(IOT_pubTopic, sub_cb)
siot.loop()
```

18.3.3 3.3 订阅多条消息

```
import siot
import time

SERVER = "127.0.0.1"         # MQTT服务器IP
CLIENT_ID = ""               # 在SIoT上，CLIENT_ID可以留空
IOT_pubTopic1  = 'xzr/001'   # “topic”为“项目名称/设备名称”
IOT_pubTopic2  = 'xzr/002'   # “topic”为“项目名称/设备名称”
IOT_UserName ='siot'         # 用户名
```

(续下页)

(接上页)

```
IOT_PassWord ='dfrobot'       # 密码

def sub_cb(client, userdata, msg):  # sub_cb函数仍然只有一个，需要在参数msg.
↪topic中对消息加以区分
  print("\nTopic:" + str(msg.topic) + " Message:" + str(msg.payload))
  # msg.payload中是消息的内容，类型是bytes，需要用解码。
  s=msg.payload.decode()
  print(s)

siot.init(CLIENT_ID, SERVER, user=IOT_UserName, password=IOT_PassWord)
siot.connect()
siot.set_callback(sub_cb)
siot.getsubscribe(IOT_pubTopic1)  # 订阅消息1
siot.getsubscribe(IOT_pubTopic2)  # 订阅消息2
siot.loop()
```

18.4 4. 借助 siot 部署智联网应用

当物联网遇上人工智能，就形成了智联网。当学生训练出一个 AI 模型，就可以通过物联网设备进行多模态交互。

1）远程感知。

2）远程控制。

CHAPTER 19

开源硬件库 pinpong 简介

19.1 1. 简介

pinpong 库是一个基于 Firmata 协议开发的 Python 硬件控制库。借助于 pinpong 库，直接用 Python 代码就能给各种常见的开源硬件编程，即使该硬件并不支持 Python。

pinpong 库的原理是给开源硬件烧录一个特定的固件，使其可以通过串口与电脑通讯，执行各种命令。pinpong 库的名称由“Pin”和“Pong”组成，“Pin”指引脚，“pinpong”为“乒乓球”的谐音，指信号的往复。目前 pinpong 库支持 Arduino、掌控板、micro:bit 等开源硬件，同时支持虚谷号、树莓派和拿铁熊猫等。

github 地址：https://github.com/DFRobot/pinpong-docs

19.2 2. 安装

可以使用使用 pip 命令安装 pinpong 库，如：

```
pip install pinpong
```

注：MMEdu 中已经内置了 pinpong 库。

19.3 3. 代码示例

示例程序以“Arduino”为例，复制粘贴代码到 python 编辑器中，并修改 Board 初始化版型参数为你使用的板子型号即可。

19.3.1 3.1 数字输出

实验效果：控制 arduino UNO 板载 LED 灯一秒闪烁一次。

接线：使用 windows 或 linux 电脑连接一块 arduino 主控板。

```
import time
from pinpong.board import Board,Pin

Board("uno").begin()
↪#初始化，选择板型(uno、microbit、RPi、handpy)和端口号，不输入端口号则进行自动识别
#Board("uno","COM36").begin()       #windows下指定端口初始化
#Board("uno","/dev/ttyACM0").begin() #linux下指定端口初始化
#Board("uno","/dev/cu.usbmodem14101").begin()    #mac下指定端口初始化

led = Pin(Pin.D13, Pin.OUT) #引脚初始化为电平输出

while True:
  #led.value(1) #输出高电平 方法1
  led.write_digital(1) #输出高电平 方法2
  print("1") #终端打印信息
  time.sleep(1) #等待1秒 保持状态
  #led.value(0) #输出低电平 方法1
  led.write_digital(0) #输出低电平 方法2
  print("0") #终端打印信息
  time.sleep(1) #等待1秒 保持状态
```

19.3.2 3.2 数字输入

实验效果：使用按钮控制 arduino UNO 板载亮灭。

接线：使用 windows 或 linux 电脑连接一块 arduino 主控板，主控板 D8 接一个按钮模块。

```
import time
from pinpong.board import Board,Pin

Board("uno").begin()

btn = Pin(Pin.D8, Pin.IN) #引脚初始化为电平输入
led = Pin(Pin.D13, Pin.OUT)

while True:
  #v = btn.value()  #读取引脚电平方法1
  v = btn.read_digital()  #读取引脚电平方法2
  print(v)  #终端打印读取的电平状态
  #led.value(v)  #将按钮状态设置给led灯引脚  输出电平方法1
  led.write_digital(v) #将按钮状态设置给led灯引脚  输出电平方法2
  time.sleep(0.1)
```

19.3.3 3.3 模拟输入

实验效果：打印 UNO 板 A0 口模拟值。

接线：使用 windows 或 linux 电脑连接一块 arduino 主控板，主控板 A0 接一个旋钮模块。

```
import time
from pinpong.board import Board,Pin

Board("uno").begin()

#adc0 = ADC(Pin(Pin.A0)) #将Pin传入ADC中实现模拟输入  模拟输入方法1
adc0 = Pin(Pin.A0, Pin.ANALOG) #引脚初始化为电平输出  模拟输入方法2

while True:
  #v = adc0.read()  #读取A0口模拟信号数值  模拟输入方法1
  v = adc0.read_analog() #读取A0口模拟信号数值  模拟输入方法2
  print("A0=", v)
  time.sleep(0.5)
```

19.3.4 3.4 模拟输出

实验效果：PWM 输出实验, 控制 LED 灯亮度变化。

接线：使用 windows 或 linux 电脑连接一块 arduino 主板，LED 灯接到 D6 引脚上。

```
import time
from pinpong.board import Board,Pin

Board("uno").begin()

#pwm0 = PWM(Pin(board,Pin.D6)) #将引脚传入PWM初始化  模拟输出方法1
pwm0 = Pin(Pin.D6, Pin.PWM) #初始化引脚为PWM模式  模拟输出方法2

while True:
    for i in range(255):
        print(i)
        #pwm0.duty(i) #PWM输出  方法1
        pwm0.write_analog(i) #PWM输出  方法2
        time.sleep(0.05)
```

19.3.5 3.5 引脚中断

实验效果：引脚模拟中断功能测试。

接线：使用 windows 或 linux 电脑连接一块 arduino 主控板，主控板 D8 接一个按钮模块。

```
import time
from pinpong.board import Board,Pin

Board("uno").begin()

btn = Pin(Pin.D8, Pin.IN)

def btn_rising_handler(pin):#中断事件回调函数
```

(续下页)

(接上页)

```
  print("\n--rising---")
  print("pin = ", pin)

def btn_falling_handler(pin):#中断事件回调函数
  print("\n--falling---")
  print("pin = ", pin)

def btn_both_handler(pin):#中断事件回调函数
  print("\n--both---")
  print("pin = ", pin)

btn.irq(trigger=Pin.IRQ_FALLING, handler=btn_falling_handler)
↪#设置中断模式为下降沿触发
#btn.irq(trigger=Pin.IRQ_RISING, handler=btn_rising_handler)
↪#设置中断模式为上升沿触发，及回调函数
#btn.irq(trigger=Pin.IRQ_RISING+Pin.IRQ_FALLING, handler=btn_both_handler)
↪#设置中断模式为电平变化时触发

while True:
  time.sleep(1) #保持程序持续运行
```

更多代码请访问官方文档。

官方文档地址：https://pinpong.readthedocs.io/

19.4 4. 借助 pinpong 开发智能作品

开源硬件是创客的神器，而 pinpong 进一步降低了开源硬件的编程门槛。pinpong 库的设计，是为了让开发者在开发过程中不用被繁杂的硬件型号束缚，而将重点转移到软件的实现。哪怕程序编写初期用 Arduino 开发，部署时改成了掌控板，只要修改一下硬件的参数就能正常运行，实现了“一次编写处处运行”。

当学生训练出一个 AI 模型，就可以通过各种硬件设备进行多模态交互。当学生训练出一个简单猫狗分类模型后，加上一个舵机，就能实现智能宠物“门禁”；加上一个马达，就能做出一个智能宠物驱逐器；加上一条快门线，就能做宠物自动拍照设备。有多少创意，就能实现多少与众不同的作品。

CHAPTER 20

Web 库 Flask 简介

20.1 1. 简介

Flask 是一个轻量级的可定制框架，使用 Python 语言编写，较其他同类型框架更为灵活、轻便、安全且容易上手，其强大的插件库可以让用户实现个性化的网站定制，开发出功能强大的网站。

文档地址：https://dormousehole.readthedocs.io/en/latest/quickstart.html

20.2 2. 安装

可以使用使用 pip 命令安装 Flask 库，如：

```
pip install flask
```

注：MMEdu 中已经内置了 Flask 库。

20.3 3. 代码示例

20.3.1 3.1 最简 Web 服务器

几行代码就能建一个 Web 服务器。

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello_world():
    return "<p>Hello, World!</p>"
```

代码是什么意思呢?

- 首先我们导入了 Flask 类。该类的实例将会成为我们的 WSGI 应用。
- 接着我们创建一个该类的实例。第一个参数是应用模块或者包的名称。**name** 是一个适用于大多数情况的快捷方式。有了这个参数，Flask 才能知道在哪里可以找到模板和静态文件等东西。
- 然后我们使用 route() 装饰器来告诉 Flask 触发函数的 URL 。
- 函数返回需要在用户浏览器中显示的信息。默认的内容类型是 HTML ，因此字符串中的 HTML 会被浏览器渲染。

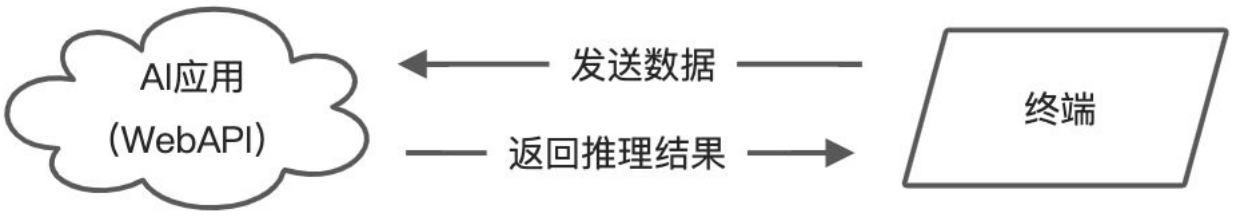
20.3.2 3.2 上传一个文件

20.3.3 3.3 一个简单的 WebAPI

20.4 4. 借助 Flask 部署智能应用

因为算力的限制，在很多应用场景中，负责采集数据的终端往往没有办法直接部署 MMEdu 或者其他的人工智能应用框架，也就是说没有办法在终端上直接推理。那么，先在服务器（或者一台算力较好的 PC 机）部署一个 AI 应用，提供 WebAPI 接口，让终端发送数据到这个服务器，服务器推理后返回推理结果。

这种做法和很多智能终端的工作原理是一样的。如小度、天猫精灵和小爱音箱等，自己都没有处理数据的能力，都要靠网络传送数据到服务器，然后才能正确和用户交流。目前中小学的很多 AI 应用，都是借助百度 AI 开放平台的。



CHAPTER 21

项目：设计“石头剪刀布”陪玩机器人

CHAPTER 22

课程：神经网络和计算机视觉实验

CHAPTER 23

课程：走进万物智联的世界

CHAPTER 24

项目：电路符号识别小助手

CHAPTER 25

常见问题解答

CHAPTER 26

预训练模型和权重文件下载

CHAPTER 27

数据集下载

CHAPTER 28

GPU 版本的手动安装

CHAPTER 29

深度学习训练参数详解

CHAPTER 30

经典数据集介绍

CHAPTER 31

图片分类模型 MobileNet

MobileNetV2: Inverted Residuals and Linear Bottlenecks

31.1 介绍

MobileNetV2 是 Google 对 MobileNetV1 提出的改良版本。MobileNets 系列的本身初衷是" **for Mobile Vision Applications**"。也就是提出一个轻量化的卷积网络模型，可以显著减少计算量和参数量，同时保持较高的准确率来提升效率，这样的模型是如此的的轻量化甚至可以在移动模型上进行训练。其主要创新性是采用了**深度可分离卷积**的思想，也就是把标准的卷积过程拆分成**深度卷积**和**逐层卷积**两个过程，大大减少了参数量。

31.2 特点：深度可分离卷积

传统的标准卷积操作在 M 通道图像的基础上，准备得到 N 个特征图，使用的基础卷积核尺寸为 $(W \cdot H)$。每一个特征图对应一个 filter，需要用 M 个小卷积核进行卷积，即每一个 filter 为 $(W \cdot H \cdot M)$ 个参数，而这样的操作重复 N 次，就得到了 N 个特征图，即总参数为 $(W \cdot H \cdot M \cdot N)$。

而为了简化这一计算过程，采用两次计算：我们首先在深度方向进行卷积（深度卷积、逐通道卷积），先改变特征图的大小而不改变特征图的数量，即输入是 M 个通道，输出也是 M 个通道，先不涉及通道数的变化：

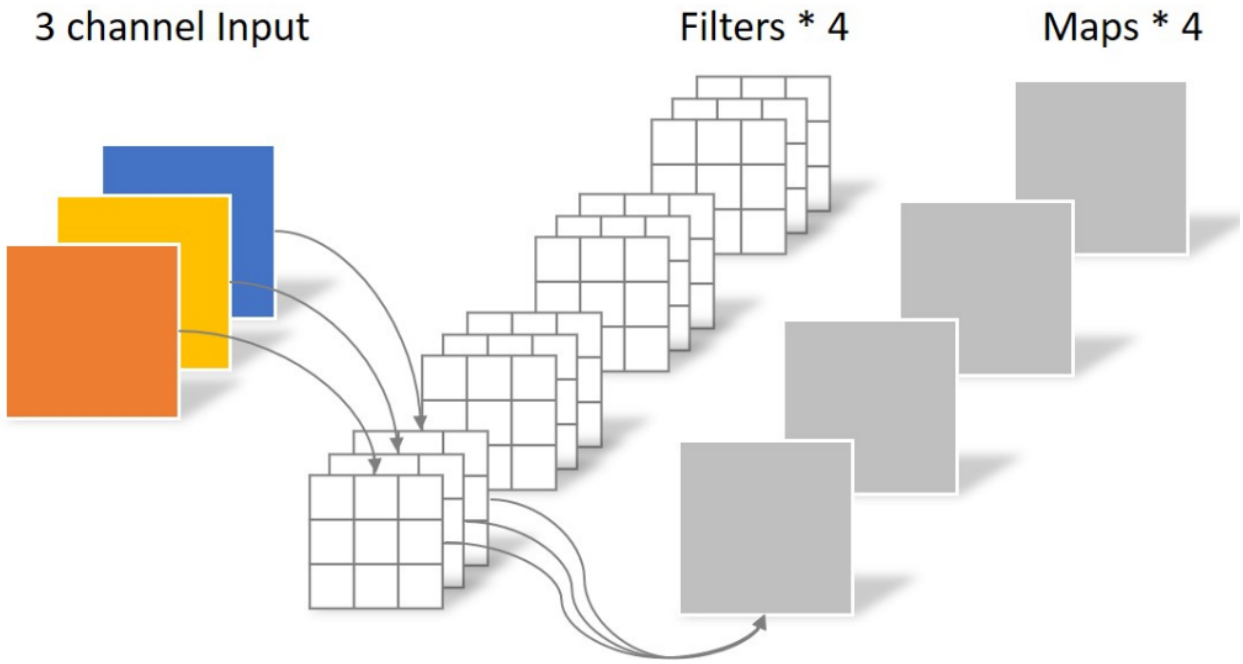
可以看到，每一个 filter 都是单层，这一步的参数量为 $(W \cdot H \cdot M)$.

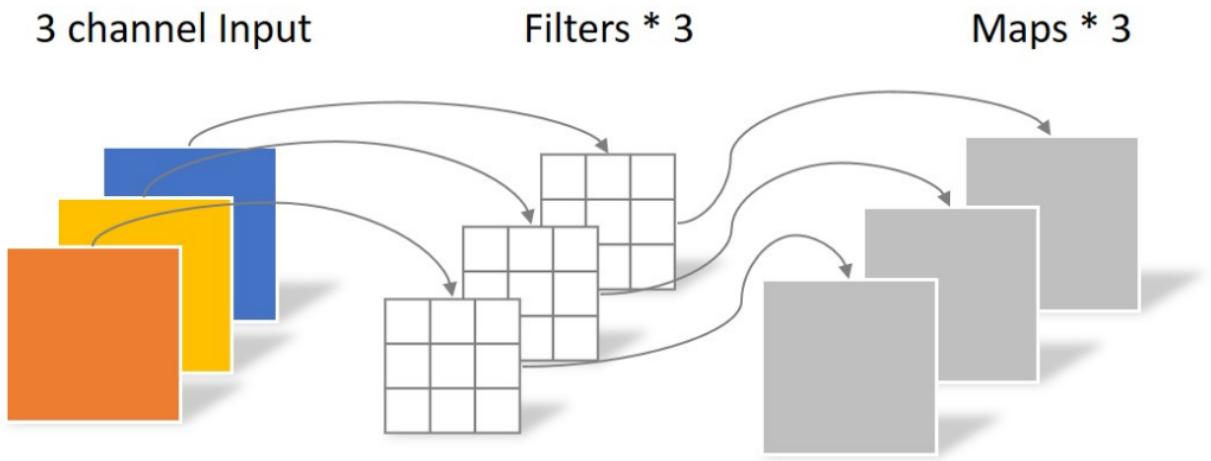
此时特征图大小已经改变，只需要从 M 维映射到 N 维即可。我们需要对每个点进行卷积（逐点卷积），添加 N 个 filters，每个 filter 尺寸为 $(1 \times 1 \cdot M)$，也就是我们常说的加权平均：

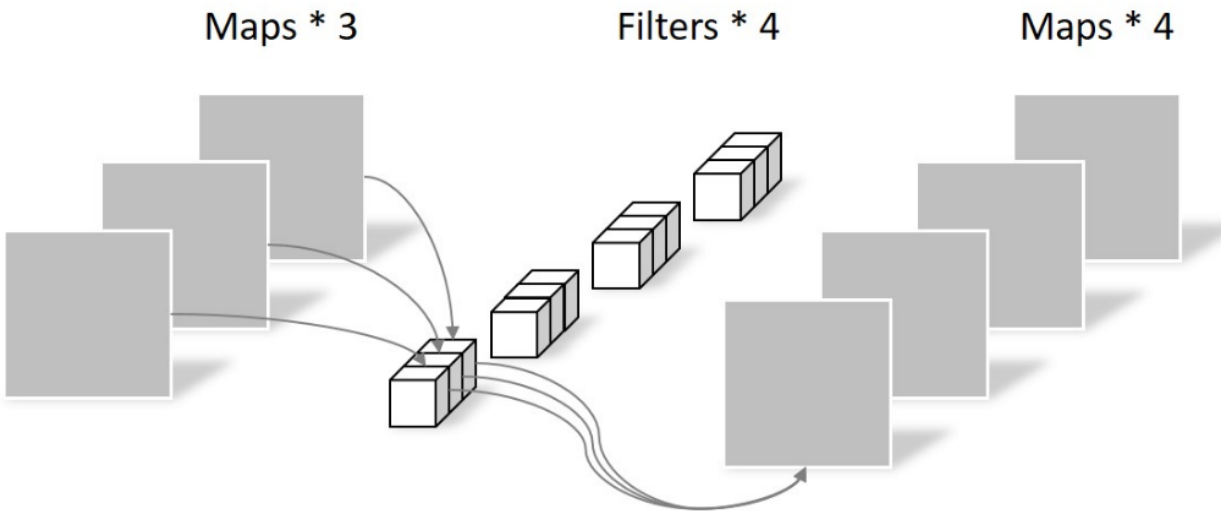
可以看到，逐点卷积通过对老通道作 N 次 "加权平均数" 的运算，得到了 N 个新的特征图。这一步的参数量为 $(1 \times 1 \cdot M \cdot N)$

这两次计算的总参数量为 $1 \times 1 \cdot M \cdot N + W \cdot H \cdot M = M \cdot (N + W \cdot H)$ 参数量是传统卷积的几分之一呢？答案是：

$$\frac{M \cdot (N + W \cdot H)}{W \cdot H \cdot M \cdot N} = \frac{1}{W \cdot H} + \frac{1}{N}$$


3 channel Input
Filters * 4
Maps * 4


3 channel Input
Filters * 3
Maps * 3



在网络中的大部分层中，我们使用的是 3×3 的卷积核，而 N 一般是几十到几百的数量级，因此一般参数可以缩减到传统方法的九分之一。而在实际测试中，原论文指出，准确度仅有 1.1% 的减少，但是参数量缩减到约七分之一：

Table 4. Depthwise Separable vs Full Convolution MobileNet

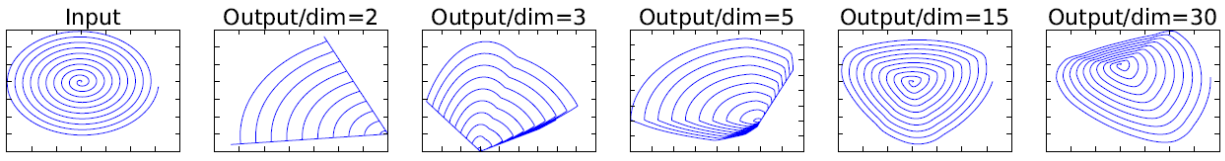
Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
Conv MobileNet	71.7%	4866	29.3
MobileNet	70.6%	569	4.2

31.3 特点：Linear Bottleneck

MobileNetV1 使用的激活函数为 Relu，这个函数也是深度学习常用的激活函数之一。在 MobileNetV2 原始论文中，MobileNet 团队指出，这个激活函数在较低维度无法保证信息的相对完整，会造成一定程度的丢失，下面这个例子有助于理解：有一条螺旋线 X 在二维空间内，我们使用某矩阵 T 将其映射到高维空间内，之后进行 Relu 操作，再使用某矩阵 T 的逆矩阵 T^{-1} 将其降维回二维。也就是说进行了一个：

$$X' = T^{-1}(Relu(T \cdot X))$$

的操作。如果没有任何信息损失，X' 和 X 就会是完全一致的，论文作者给出的结果是：



可以很直接的看出，维度越高，Relu 造成的损失越小，而在低维度的情况下，信息失真很严重，这样会造成很多卷积核的死亡（权重为 0）。于是作者在某些层舍弃了 Relu，采用了线性的变化函数，进行了一次升维的操作，作者将其称之为“Linear Bottleneck”。此部分的细节我们不做深入阐述，这一步的作用就是给原网络升维，从而避免了很多卷积核的死亡。

31.4 网络结构

至此我们给出网络结构：

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

可以从网络结构中看到上面我们描述的深度卷积层（conv2d）和逐点卷积层（conv2d 1x1）。网络结构肉眼可见的简洁和清晰，而效果也不俗。

31.5 优点

总结一下优点，最大的优点就是创新性的可分离卷积带来的大量参数减少，从而导致网络的轻量化。此外，V2 比起 V1，还增加了 Linear Bottleneck 机制来避免卷积核的死亡，从而提高参数利用率。

31.6 使用领域

- 算力相对较低的移动平台的模型部署
- 减少大型项目的运行时间同时保留较高的准确率

31.7 参考文献

```
@misc{howard2017mobilenets,
      title={MobileNets: Efficient Convolutional Neural Networks for Mobile Vision
↪Applications},
      author={Andrew G. Howard and Menglong Zhu and Bo Chen and Dmitry Kalenichenko
↪and Weijun Wang and Tobias Weyand and Marco Andreetto and Hartwig Adam},
      year={2017},
      eprint={1704.04861},
      archivePrefix={arXiv},
      primaryClass={cs.CV}
}
@misc{sandler2019mobilenetv2,
```

(续下页)

(接上页)

```
      title={MobileNetV2: Inverted Residuals and Linear Bottlenecks},
      author={Mark Sandler and Andrew Howard and Menglong Zhu and Andrey Zhmoginov␣
↪and Liang-Chieh Chen},
      year={2019},
      eprint={1801.04381},
      archivePrefix={arXiv},
      primaryClass={cs.CV}
}
```

CHAPTER 32

图片分类模型 LeNet-5

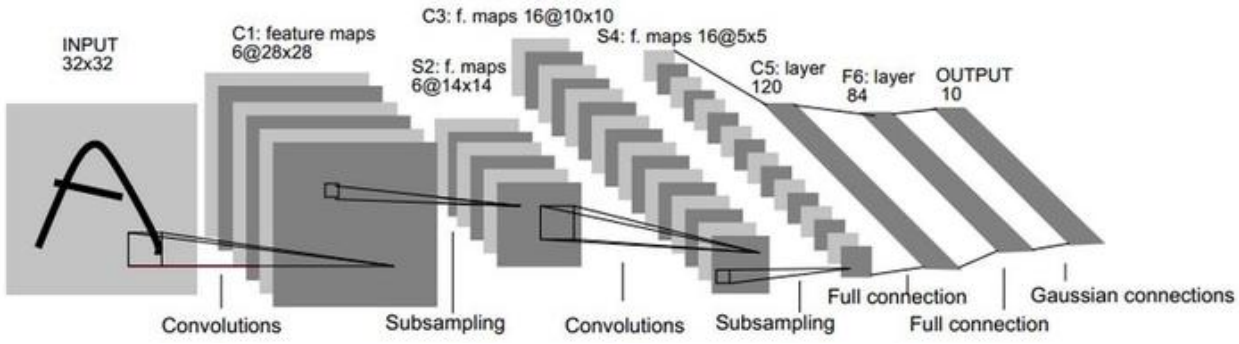
Backpropagation Applied to Handwritten Zip Code Recognition

32.1 简介

LeNet 是一种用于手写体字符识别的非常高效的卷积神经网络，通常 LeNet 指 LeNet-5。

32.2 网络结构

LeNet 的网络结构示意图如下所示：



即输入手写字符图片经过 C1，C2，C3 这 3 个卷积神经网络后，再经过两层全连接神经网络，输出最终的分类结果。

32.3 优点

- 简单，易于初学者学习与使用
- 运行速度快，对硬件设备没有要求

32.4 适用领域

- 手写体字符识别

32.5 参考文献

```
@ARTICLE{6795724,
  author={Y. {LeCun} and B. {Boser} and J. S. {Denker} and D. {Henderson} and R. E.
↪{Howard} and W. {Hubbard} and L. D. {Jackel}},
  journal={Neural Computation},
  title={Backpropagation Applied to Handwritten Zip Code Recognition},
  year={1989},
  volume={1},
  number={4},
  pages={541-551},
  doi={10.1162/neco.1989.1.4.541}}
}
```

CHAPTER 33

Indices and tables

- genindex
- modindex
- search