**显卡驱动：**

下载地址：<https://www.nvidia.cn/Download/index.aspx?lang=cn>

安装版本要与显卡型号对应

查看显卡版本、支持的最大cuda版本(不是当前cuda版本)、GPU使用情况

**cuda：GPU驱动程序**

下载地址：<https://developer.nvidia.com/cuda-toolkit-archive>

安装版本要与显卡驱动对应(nvidia-smi查看，显卡驱动版本向下兼容https://docs.nvidia.com/cuda/cuda-toolkit-release-notes/index.html)。cuda可以安装多个并切换版本，安装和卸载最好搜索一下教程。windows安装可以放到非C盘并创建子文件，安装包下载后要解压，注意解压位置要单独创建不要有其他文件，因为cuda安装完后整个文件夹会全部自动删除

完整的cuda包括cunda驱动和cuda工具cuda toolkit，不需要cuda toolkit也能使用cuda，但无法使用nvcc -V。官网下载cuda会自动带上cuda toolkit

nvcc -V：查看cuda版本

**cudnn：**神经网络加速库

下载地址：https://developer.nvidia.com/rdp/cudnn-archive

安装版本要与cuda对应。下载的cudnn压缩包中的bin、include、lib(linux版没有bin)是所需要的文件。windows复制放到cuda的development中的bin、include、lib中。linux同样放到对应的cuda文件下，如果使用时提示没找到dll，则使用【sudo ldconfig lib位置】添加到环境变量中。卸载时删除其中cudnn的文件即可

**conda：**适用于任何语言的软件包、依赖项和环境管理系统

安装miniconda3或anaconda3会自带conda。安装之后要重启

conda -V：查看conda版本

conda info -e：查看所有环境

conda activate 环境名：进入环境

conda create -n 环境名称 python=3.X：创建环境

conda env remove -n 环境：删除环境

conda install python==版本号：更换python版本

conda clean -a：删除从未使用的库、索引缓存、tar包

**Miniconda3**

下载网址：https://mirrors.tuna.tsinghua.edu.cn/anaconda/miniconda/

比Anaconda3小很多，相当于简化版的Anaconda3

linux安装时使用’sh 安装包位置.sh’安装，之后一直按回车，再输入yes。安装后需要重启服务器才能使用conda

**Anaconda3**

下载网址：https://mirrors.tuna.tsinghua.edu.cn/anaconda/archive/

python环境管理软件，通常python的库和包都安装在Anaconda3下，使用编程软件如pycharm时直接选用Anaconda3中的python.exe

适用于windows和linux等

**Anaconda3 Jupyter Notebook**

Jupyter在网页中打开，在页面中输入程序运行时结构会一直存在，如果打开多个Jupyter页面，运行一个页面关闭后还会一直占内存，除非关闭整个程序

Anaconda3\Lib\site-packages:安装的库位置

Jupyter:后缀.ipynb

Pycharm:文件路径和Anaconda3一致，Python3路径使用Anaconda3的

Jupyter中使用history查看所有运行记录

Jupyter中cell>>All Output>>Clear只清楚记录

conda--version:查看anaconda版本

XX.\_\_version\_\_:查看版本

XX.\_\_path/file\_\_:查看文件位置(file位置更细)

**Anaconda Prompt**：anaconda版的终端命令框

python:查看python版本并进入python编译。exit()/CTRL+C:退出环境

pip/conda --version:查看版本

pip/conda list：查看所有安装的包

**Pycharm**

环境推荐使用anaconda3中的环境，在文件->设置->项目->python interpreter中选择anaconda3下的python.exe

左下的terminal是pycharm的命令终端

简单的语法错误会显示红色波浪线，拼写函数时会智能的跳出存在的函数

代码左侧打上断点可调式，右键可拖动断，运行到断点处时并未执行程序

在pycharm中更改文件名时可以启用引用搜索，同步更改代码相关文件名

pycharm中VCS菜单->git->clone复制远程库地址可以克隆项目

pycharm每次启动时都会更新环境和库的路径信息

Ctrl+Alt+L：使整个页面的程序格式规范。Pycharm有规范的代码格式，格式不规范时会出现波浪线(但不影响运行)。可能与QQ的快捷键相冲突

CTRL+F：在当前文档中搜寻关键词，可按↑↓按钮快速寻找

CTRL+B：快速转到函数所在位置

**pip工具：**系统一般默认已安装pip，用于管理和维护python包。当程序中调用库时可能无法更换库。更换库版本时需要先卸载原版本或指定版本

pip下载：<https://pypi.org/project/pip/>

解压后从命令终端进入文件夹使用python setup.py install，删除安装包

pip --version：查看pip版本

pip install 库(==版本号)：安装库。会自动寻找和系统对应的型号，不指定版本时安装最新版本。已安装该库后，指定版本再次安装可更换版本

（-i https://pypi.doubanio.com/simple：添加豆瓣源）

（-i https://pypi.tuna.tsinghua.edu.cn/simple：添加清华源）

pip uninstall 库：删除库(包括pip)

如果直接pip安装失败，可以去官网下载要安装的版本的.whl文件到本地，然后使用pip install 文件位置.whl

使用VPN时无法使用pip命令

pip install numpy pandas matplotlib tqdm opencv-python albumentations pyyaml wandb onnxruntime-gpu onnx-simplifier pycocotools -i https://pypi.doubanio.com/simple

**Python3**

conda install python：windows中升级python

sudo apt install python：linux中升级python

**基本知识：**

程序运行时.py→python解释器→.pyc。再次运行时若源代码没变python解释器直接运行.pyc，若源代码改变先生产.pyc再运行

python中有一些自动优化的算法，比如B=A或A传入函数为B时，B只是引用A，只有对B做改变时，才会额外创造一份B的内存

python库有串行计算和并行计算。速度通常GPU并行>CPU并行>CPU串行

不需要定义常量或变量，单双引号没有区别，句子结尾不需加’;’，函数结尾加’:’

变量自带内置属性(不同类型的变量自带属性不一样)，在Pycharm中使用调试时可以看到，比如tensor.device.type：查看张量所在的设备

注释：#内容。CTRL加’/’快速注释选中的段。’’’内容段’’’

只有变量没有常量，但习惯把不改变的变量用大写表示

若列表list=[1,2,3]，可以直接用a,b,c=list进行一一对应赋值，numpy数组等其他数据结构同理

下标从0开始，A[2]为A第三个元素，A[0:2]表示A[0]和A[1]，左闭右开

部分python库是由c++写的，只是可以通过python调用，库中的python文件的函数只是个占用的空壳，只有个pass

python是进程安全的，全局变量在多进程中修改不影响

python的部分库需要c++环境，可以安装VS

**导入库：**

import 库：在当前路径下导入库。部分库在运行时会在用户目录下生成‘.库名’的临时文件，甚至下载一些模型文件

库.文件夹.文件(py).类/函数：调用库中的某个函数。如果是用户自己写的文件程序，一般只能from 库.文件夹.文件(py) import 类/函数

from 库 import ....函数：调用库中的某函数

调用库时如果传入路径等内容，是以最开始运行的文件所在目录为相对目录

有的库分CPU/GPU版，CPU版要小很多，但GPU版在CPU和GPU上都可以用

**元组tuple：**tuple1=(1,2,’A’),可不加括号，如果是单独数字要加逗号区分，如(1,)。静态数组，元素不可变，元组定义后不可再对其中的元素修改

tuple1=(1,)//tuple2=(2,)//tuple3=tuple1+tuple2：tuple3=(1,2)

tuple2=tuple1\*3：tuple2=(1,1,1)

**列表list：**list=[1,2,”A”]，动态数组，内存中不连续存放，读写效率较低，占内存大。列表中存放的是对应数值的内存地址。列表中可以存放元组、字典、np数组、tensor张量等。列表可以应用list[i:]，但不可以批量赋值list[i:]=...

list(dict)：将字典的键转化为数列表

list2=list1：只是对list1引用，列表内存地址一样，对列表更改时同步

list2=list1.copy()/list1[:]：单独分配list2的内存空间，将list1内容复制给list2(list1可为多级列表)

list2=[i for i in list1]：同.copy()

list=[i... for i in list]：对list中每个元素进行操作

list[a][b]：引用，不能用[a,b]。list[0:3]：左闭右开

list[::i]：间隔为i显示。list[-i]：倒数第i个

list[-i:]：最后i个元素组成的列表

list[a:]：第a+1个元素之后所有元素组成的列表，a超过边界时为空列表

list.append(A)：增加一个元素A

list.extend(A)：将A中每个元素分别增加进去

list.remove(A)：删除列表中元素A,如有相同元素，则只删除一个元素A

list.pop(a)：删除第a+1项,a=0,1,2...；list.pop():删除最后一项

list.reverse()：倒序排列

list.sort()：按升序排列

sorted(list,key=fun1,reverse=False)：排序。如果list是二维则排序根据一维的第一个数大小，key是根据fun1(list)的值排序,reverse=True倒序排列

function1=lambda x:功能：匿名函数。使用：function1(a)时a传入到x中，并执行后面的功能

list.count(A)：统计A在list中出现的次数

list=[0 for i in range(a)]：创建长度为a的全0列表

list=[[]for i in range(a)]：创建二位列表

map(function1,list)：将list中每个元素执行function1，返回map格式。function1可以为int、lambda等

list(map(int,list))：将列表中字符的转化为数字

random.shuffle(list)：将元素打乱，会直接更改list，不需要返回

enumerate(list,start=0)：将列表中元素扩充索引。enumerate([A,B])得到[(0,A),(1,B)]

‘A’.join(list)：list元素都为str时，将每个元素用A连接成一个str

**集合set:**集合中元素不能重复,set1={1,2,’A’}；set1=set(‘12A’)，<set1={‘1’,’2‘,’A’}>，set1=set((‘12’AB’))，<set1={‘12’,’AB’}>

set1.add(A)增加元素A,set1.update(‘AB’)分别增加字符A和B,set1.update不可增加数字

set1.remove(A),删除元素A,若A不存在会报错,set1.discard（A）,删除元素A，若A不存在不会报错,set1.pop()删除第一个元素,set1.clear()删除所有元素

set3=set1-set2求差集,|求并集,&求交集,^求非交集

**字典dict:**由键和值组成,字典的键不能重复,字典一般嵌套在列表里用列表操作，字典可以嵌套。字典是无序的

dict={}：创建空字典。dict={A:[A,1],B:2}：创建字典

dict[A]查询A键对应的值，若无A键则报错

dict[A]=[]：A键对应的值为列表

dict[A][0]：查询A键中的第一个值

dict[A]=B：改变键A对应的值，没有A则创建一个A并赋予键值B

dict.keys()：所有关键字，返回的值要引用需tuple/list转化一下

'键名' in keys()：判断是否存在键并返回True或False

dict.values():所有值

dict.items()：将字典中的键和值全部列出。如[(A,1),(B,2)]

len(dict)：查看字典中的键数

dict1.update(dict2)：添加字典dict2，没有返回值

**字符串str：**定义后不能对其中的元素更改，只能重新定义

str2=str1：不同于列表，字符串可以直接赋值

str(a)：将a转化为字符；str1+=’A’：添加元素到字符串中

str1[-i]：倒数第i个字符；str1[-i:]：取出最后i个字符

str1[a:b]：按下标截取字符串

list(str1)：把字符串拆分为单个字符并组成列表

str1.split(‘A’)：以str1中A划分字符返回列表；‘A’不填等于空格

str1.strip(‘A’)：去除字符串两边的A字符。‘A’为空时可去除’\n’

str1.upper():全转化为大写。str1.lower()：全转化为小写

str1.replace(‘A’,’B’,a)：将字符串中的A替换为B，a为替换前a个匹配的A，不指定a则全替换

‘12{}34{}56’.format(A,B)：将A，B加入字符串中

A.join(B)：在B的每个字符间插入A。A为分隔符，如’ ’，B为列表、字符串、元组等

str1.rjust(a,’A’)：在str1左边填充A达到a长度

str1=’function1’//eval(str1)：将字符串变为表达式，执行function1

**字节类型：**

bytes1=str1.encode(‘UTF-8’)：将字符串转换为字节类型。若str1=’abc’，则str1.encode(‘UTF-8’)为b’abc’

str1=bytes1.decode(‘UTF-8’)：将字节类型转换为字符串

**读/写文件：**

f=open(文件路径,读取模式,encoding='utf-8')：打开文件，如果文件不存在则会创建新文件。读取模式’r’读取为字符串，’rb’读取为字节类型，图片读取选用，’w’清空重写(如果该文件不存在会创建一个空的文件)，’a’接着最后一行写入数据；encoding='utf-8'为读取编码方式。结束要使用f.close

with open(文件路径/新文件路径,读取模式,encoding='utf-8') as f：等同于f=open()，结束之后会自动调用f.close()，如果忘记用close会占用系统资源

f.read(i)：读取i个字符内容

f.read()：读取整个文件。对于大文件不要使用

f.readline()：读取一行，包括\n。\n可加.strip()去除

[\_.strip() for \_ in f.readlines()]：将txt文件中所有行变为列表元素

f.readlines()：读取所有行到列表中，包括\n。如果最后一行什么都没有则不会读取，转行符号算作上一行的内容

f.write(str1)：将字符串写入文件。如要转行可写入转行符’\n’；如果要写入列表，要先将列表序列化f.write(json.dumps(list))，或直接用json.dump(list,f)便捷写入。对于元素都为list的列表，使用f.write('\n'.join(label\_list)+'\n')快速写入

f.writelines(list)：将list中元素合并后一起写入。如果要把每个元素作为一行，要在每个元素后加上\n

f.truncate()：清空文件

**输入与输出：**

input(‘提示信息’)：输入一个字符串，按回车确认(回车不会输进去)

print(A,end=’’):连续输出不换行

print(f‘qwe{A}asd{B}’)：将待输出的值A、B插入输出信息中

print(‘qwe{}asd{3:.4f})’.format(A,B))：将A,B对应填入{}中。3:.4f表示保留3位整数和4位小数

**条件语句：**

系统会先运行and/or前面的条件语句，如果此时已经可以得出判读结果，就不会再运行后面的条件语句了。False、None、0、[]、{}等为否，True、非零值、非空值等为是

if A (not) in list:：判断A是否在list中，优先级not>and>or

if 条件1 end(or) 条件2://...//elif ...://...//else://...

x=a if 条件 else b：满足条件时x=a，否则x=b

for i in [1,2,3]:：for遍历

for i in range(10):：遍历0,1,..,9。等同于(0,10)、(0,10,1)。(9,-1,-1)表示从9、8一直到0

for i,j in zip(a,b):：并行遍历

for index,item in enumerate(list):：同时遍历索引和值

while A<B：while循环。ctrl+d强行结束循环

**函数、类与调用：**

函数需先定义后调用，def需写在调用前面。但使用类时程序运行前会提前初始化，可以写在调用后

def 函数名(参数1=默认值，参数2=默认值...):：定义一个函数，若参数没有赋值则使用默认值，默认值要从右依此往左给,如果没有return返回None。

global A：函数的全局变量A可以在函数中使用，要在函数中加

@function1//def function2(...)：装饰函数。将fun2作为fun1的参数，先执行fun1，再执行fun2

class 类名(继承的类)：定义一个类。类的对象可以是函数或变量。调用方法：‘类名.函数(参数)’；有\_\_init\_\_初始化时：‘类名(\_\_init\_\_参数).函数(参数)’，定义了\_\_call\_\_时：可以使用‘类名(\_\_init\_\_参数)(参数)’。如果类中定义了函数 self.data，可以直接使用class.data得到值，也可以使用class.data=a赋值

class1 类名(object)：继承object类可以有更多的功能。常用class1 继承object，class2继承class1嵌套

def \_\_init\_\_(self,A)//self.A=A：初始化类（可不用）。初始化类可以让参数self.A在类中共享。self.函数名()：调用类中的其他函数

def \_\_call\_\_(self,A)：可以使用’类名(\_\_init\_\_参数)(参数)’调用\_\_call\_\_函数

class2 类名(class1)//def \_\_init\_\_(self,class1的参数)//super().\_\_init\_\_(class1的参数)：将继承类中的参数初始化到此类中(运行时会跳转到继承类中的初始化并执行一遍)

def \_\_call\_\_(self,B)：可以直接调用类。class1(A)(B)

def \_function1(self,B)：初始化后类中定义函数第一个参数是self，自动从初始化中传入参数，B为调用需时传入的参数，如class2(A).\_function1(B)

\_\_xx\_\_是内置变量，每个python模块(.py)都包含\_\_name\_\_：当前模块名称

if \_\_name\_\_==’\_\_main\_\_’：如果是主程序则运行，一般在库中加做测试。同时main是python的最高命名空间，在这里面的代码不会被子进程重新读取

**其他：**

round(a,decimals=0)：四舍五入，结果为浮点数。decimals为保留小数点后位数，0为保留到个位，-1为保留到十位

A,B,C=a,b,c:连续赋值；A,B=B,A：python特有交换数值

abs(a)：求a的绝对值

ord(‘A’)：字符转为ASCLL码。chr(‘A’)：ASCLL码转字符

try://...//except://...：异常检验

len(a\*b\*c的列表/数组)得到第一维a

字符与ASCLL码对应：0-9:48-57；A-Z:65-90；a-z:97-122

\_\_init\_\_.py 文件的作用是将文件夹变为一个Python库，Python 中的每个库中文件夹下都有\_\_init\_\_.py 文件。没有\_init\_\_.py 文件时只能import 文件夹.文件 来导入函数，不能使用import 文件夹//文件夹.文件。

assert 条件,’提示’：检查条件是否错误，如果错误停止程序

isinstance(A,type)：判断A的数据类型是否为type，返回布尔值。type为单个类型如int或集类型合如(int,str,list)

with A() as B:：相当于B=A()，只是with执行完后会自动关闭B

del A：删除变量A

/最上级目录。./：当前目录。../：上级目录

**数据存储方式：**

id(A):查看对象A的内存地址

python采用分离式结构动态数组

不同的对象内存地址不一样:id(A)!=id(B)；每个对象中具体的某个数值在系统中只有一份，保存的内存地址固定: A=[1]，B=[1]，id(A[0])=id(B[0])

对于任意对象，创建时随机分配一个合适的内存地址，系统有一个计时器记录该对象被引用的次数，每次有新的对象引用该对象，计数器+1，用del释放一个引用时-1，当垃圾回收机制发现某对象引用次数为0时将其内存删除

创建N个元素的列表时分配N+1个元素的内存，第一个元素存储列表长度等信息，使用list.append()会按一定规律额外创建更大的内存空间（之后使用append时不再创建，如果还是超过了，又继续额外分配更大的内存空间），将list的数据复制到新列表中，并消除旧列表。但list的内存地址不变

**多线程：**

https://blog.csdn.net/weixin\_61805348/article/details/125021831?ops\_request\_misc=%257B%2522request%255Fid%2522%253A%2522166598686216782425148099%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request\_id=166598686216782425148099&biz\_id=0&utm\_medium=distribute.pc\_search\_result.none-task-blog-2~all~top\_positive~default-2-125021831-null-null.142^v58^pc\_rank\_34\_queryrelevant25,201^v3^control\_2&utm\_term=PYTHON%E5%A4%9A%E7%BA%BF%E7%A8%8B&spm=1018.2226.3001.4187

**import numpy as np**

Numpy是python中最基本的库之一，串行计算但速度也较快

Numpy包中数据结构：数组、矩阵。ndarray：内存中连续存放，CPU中计算，默认float64和int32

array.shape：查看形状。array.dtype：查看数据类型

array.size：查看元素数。array.ndim：查看维度

array=array.astype(np.float32)：转换np数据类型

**创建np数组：**

np.array([[1,2],[3,4]])：创建数组([[1,2],[3,4]])

np.arange(a,b,间距).reshape(a,b)：创建数组([a,1,...b-1])

np.linspace(a,b,c)：从a起始到b结束，创建c个间隔均匀的数组

np.zeros((a,b),dtype)：创建a行b列全0数组。dtype=bool时创建全为False的数组

np.ones((a,b),dtype)：创建a行b列全1数组。dtype=bool全True数组

np.full((a,b),value,dtype)：创建a行b列全value数组

np.random.seed(种子号)：设置随机种子,相同种子的随机值一致

np.random.rand(a,b)：创建a行b列随机0-1均匀分布的数组

np.random.randint(a,b,(c,d))：创建形状为(c,d)值为a到b的随机整数

np.random.choice(array,size,replace)：从array中随机抽取元素组成形状为size的数组。replace=True/False表示抽取元素是否可以相同

np.random.normal(loc,scale,size)：创建正态分布数组(数值很小，之间)。loc为均值；scale为标准差，scale越大效果越明显，可为0.05；shape为形状

array.item()：如果取出array对象中的

**基本操作：**

array.shape：显示数组形状

array.dtype：显示数组类型如float64。str(array.dtype)转化为字符串

array.tolist()：将array变为列表，不能用list(array)

array.copy()：复制一个array

np.random.shuffle(array)：打乱array中各元素的顺序，会直接改变array中的元素，没有返回值

array[array==a]=b：将数组中元素a替换为b

bool\_opposite=~bool：布尔数组取反

array[a,b]：第a+1行b+1列 | array[a][b]：先取[a]再取array[a]的[b]，某些情况下与前者有区别

array[a:b,c:d]：第a+1到b行c+1到d列 | [[a,b,c,d]]：引用多个数

array[...,a]：最后一维第a+1个元素。如果不知道数组是一维还是二维时，用array[...,a]而不用array[a]/array[:,a]

array[0,i]：会得到(1)的形状。array[0,i:i+1]：会得到(1,1)的形状

array2=array1[np.newaxis]：增加维度。[np.newaxis]在0维增加；[:,np.newaxis]在1维增加；[...,np.newaxis]加在最后

np.around(array,decimals=0)：四舍五入，结果为浮点数。decimals为保留小数点后位数，0为保留到个位，-1为保留到十位

np.append(array,[...],axis)：添加元素

np.where((array<a)&/|(array>b),A,B)：返回与array相同形状的数组。并行遍历array中所有元素，满足条件为A，否则为B。存在多个条件时，各条件遍历次数需一致，条件中的嵌套函数，但注意函数是用所有数据计算，只是比较的时候才逐个遍历。A,B可以为函数，函数可以执行，返回的元素为None。效率高，常用于数组元素的筛选

np.argmin/argsmax(array,axis=0)：取array最小值/最大值下标的索引

np.clip(input,min,max)：将input中的值压缩到min到max之间

np.sort(array,axis=0)：根据axis维度的值从小到大排序

np.argsort(array,axis=0)：根据axis维度的值从小到大排序，返回下标

np.round(a,decimals=0)：近似数，结果为浮点数。decimals为保留小数点后位数，0为保留到个位，-1为保留到十位

np.ascontiguousarray(array)：将数组的内存变为连续存储

**数据转换：**

np.frombuffer(字节类型数据,dtype=np.uint8)：从字节类型中解码数据为np.array(一行)

array.tostring()：将np数组(一行)转换为字节类型

**维度变换：**

array.reshape(a,b)：改变数组形状，如果array(a,b).reshape(a,b,1)表示变成三维。reshape(c,-1)表示变成c行，列数自动分配

array.flatten()：将数组变成一维

np.append(array1,array2,axis=0)：将两个数组在axis维合并

np.stack([array1,array2],axis)：在axis维度上叠加数组，增加维度

np.vstack([array1,array2])：上下堆叠数组，不增加维度

np.hstack([array1,array2])：左右堆叠数组，不增加维度

np.concatenate((array1,array2),axis)：在当前的axis维度叠加数组，不增加维度。axis=0时相当于np.vstack，axis=1时相当于np.hstack

np.delete(array,(a,b,...),axis=0)：删除axis维上下标为a...的元素

array.repeat(a,...b,c)：复制c个array到0维度，并在1维度将结果复制b个，并在更高维不断重复直到第一个数a

array.transpose(0,2,1)：更改维度，0,2,1表示将原来的1,2维换位

np.stack([A,B]).reshape(-1)：实现A、B数组中元素交叉

**计算：**

np.diff(array)：求array差分

np.argmax(array)：求array中最大元素的索引

np.square/exp(array)：求每个元素的平方/e的平方，返回数组

np.maximum/minimum(A,B)：逐位比较A,B中的每个元素，保留最大/小值

np.max/min/sum/mean/std(array,axis)：求axis维度的最大值/最小值/和/均值/方差(不指定axias时求所有值的)。若array形状为(2,3,4)，axis=0，则结果形状为(3,4)，若axis=1，则结果形状为(2,4)

**矩阵运算：**

array.T：转置

np.linalg.inv(array)：求逆

np.dot(a,b)：求a和b的点积(矩阵相乘)

**函数拟合：**

np.polyfit(x,y,deg)：拟合曲线。deg=1,2...为拟合多项式阶次

np.polyval(model,x)：用模型求出预测值

**np格式保存：**

np.savetxt(path+name,array)：保存1维/2维数据到TXT文件

np.savez(path+name,array)：保存多维数组到np专用npz文件中

np.load(path,array)：读取npz文件（numpy专用二进制文件）

**import pandas as pd**

Pandas是python中最基本的库之一，串行计算

Pandas包中数据结构：序列series、数据框dataframe。

**pd.Series(dicet1)：**将字典转化为序列,序列由索引和值组成

pd.Series([1,’A’],index=[2,’B’])：index为序列的索引;若index没指定,则自动生成索引0,1

Series1.sort\_index；按索引升序查看所有序列

Series1.sort\_values:按值升序查看所有序列

Series1.index：查看所有索引。Series1.index[a]：查看序列a+1的索引

Series1.values：查看所有值。Series1.values[a+1]：查看序列a+1的值

Series1[index1]：查找索引index1对应的值

Series1.append(pd.Series([...],index=[...]))：增加一项

Series1.drop(index1)：删除索引index1对应的序列

Series1[index1]=A：将索引index1的值改为A

**pd.DataFrame(dict)：**数据框由索引、列名、内容组成;将字典转化为数据框,字典的键对应列名,值(列表、np数组)对应内容,索引从0开始自动生成。数据框的0维度是内容数，1维度是列名数

**pd.DataFrame(list1,columns=list2)：**将列表/数组转为数据框。若list1是一个一级列表，则list2中只有一个列名，list1作为一列；若list1是二级列表，则list2列名数应等于list1列数

**pd.DataFrame(np.arange(a).reshape(b,c),columns=[A,B...])**：创建df。pd.DataFrame(columns=[A,B...])创建空的df

pd.read\_csv(r‘文件位置.csv’,header=0,dtype=np.float64)：导入csv。前面加r是为了把文件位置中的\转化为/，否则会报错；header=0时会将第一行读取作为表头，=None时不需要表头。但linux中不能用r’\’

pd.read\_excel(r‘文件位置.xls(xlsx)’,sheet\_name):导入表格。sheet\_name指定读入的工作表，’sheet1’读取’sheet1’为df，[A,B]读取A、B为字典{A:df,B:df}，None读取所有的工作表为字典

df.to\_csv(r'A.csv',index=True,header=True)：保存为scv文件。index=True时会把数据框的索引作为第1列保存到csv中；header=True时会把列名保存到csv中

df.to\_excel(r'A.xls/xlsx')：保存为xls/xlsx文件

df2=df1:df1和df2都指向一个数据框。df2=df1.copy():复制一份给df2

df.columns：列名 | df1.columns=(A,B...)：更改列名

df.index：索引

df.values：将数据结构转为numpy数组

df[A]:选择列(结果为序列) | df1[A]=[...]：更改/添加列

df[[A,B]]:选择多列(结果为数据框)

df.loc[a]：选择索引为a的行(结果为序列)

df.loc[a:b]：选择索引为a:b(包括b)的行(结果为数据框)

df2=df1.drop([A,B...],axis=0)：axis=0时删除行，axis=1时删除列。不会改变原来的df1

pd.concat([df1,df2],axis,...,ignore\_index)：合并df1和df2;axis=0时按行合并,列名要一致,ignore\_index=True时索引按顺序合并;axis=1时按列合并,索引要一致

df.stack().unstack(0)：数据框转置

**时间处理：**

DatetimeIndex=pd.DatetimeIndex(array)：创建DatetimeIndex对象

//DatetimeIndex.second/minute/hour/day/month/year/dayofweek：取出DatetimeIndex中某一时间的值

**import matplotlib.pyplot as plt**

plt.title(‘标题’)：创建图像标题。重新创建时会清空原有的操作，相当关于重新创建一张空白图

plt.xlabel(‘A’)/plt.ylabel(‘B’)：添加x/y轴标签

plt.plot(y,color=’green’,label=’B’)：画二维图，x轴自动取等间距

plt.savefig(保存的文件夹位置.jpg)：保存结果。要放到plt.show()前否则为空白

plt.show()：显示图片到窗口

Plt.close()

**import tensorflow as tf**

(pip install tensorflow：CPU版本)

(pip install tensorflow-gpu：GPU版本，要有对应的cuda版本)

与cuda版本对应：<https://www.tensorflow.org/install/source#gpu>

默认动态图机制，tf.tensor引用方式和数组一样

tf.tensor张量：可以运行在GPU和TPU上，小数默认32位浮点，当运行在CPU时和numpy数组共用内存，运行在GPU时从内存中复制一份到GPU上，多维张量在内存中以一维数组存储

tf.config.experimental.list\_physical\_devices(‘CPU’/‘GPU’)：查看CPU/GPU

tf.config.experimental.set\_memory\_growth(GPUS[0],True)：允许运行时分配更多内存

tf.constant(value,dtype):创建tf.tensor常量张量，value可为数字、列表、numpy数组。dtype=tf.float/int...；张量显示时带<>

tensor.numpy():将tensor张量变为numpy数组

tf.shape(a):看形状；tf.size(a):看元素数；tf.rank(a):看维度

tf.cast(tensor,dtype=tf.float32)转换数据类型

tf.reshape(tensor,(m,n,k...)):转换tensor形状

tf.zeros(shape,dtype):全0张量 | tf.ones(shape,dtype):全1张量

tf.fill(shape,a):全a张量

tf.random.normal(shape,mean,stddev):根据正态分布随机初始化值。shape是初始化形状，mean是均值默认0；stddev是标准差默认1

tf.reshape(tensor,shape):改变形状

tf.expand\_dims(tensor,axis):增加维度

tf.gather(tensor,axis,indices):按索引值提取张量

tf.equal(A,B)分别比较两个矩阵各数是否相等,返回True/False矩阵

tf.matmul(A,B)将两个矩阵相乘

tf.add(A,B):逐元素相加。tf.subtract(A,B):逐元素相减

Variable1.assign\_sub(A):逐元素相减，更新梯度时使用

tf.multiply(A,B):逐元素相乘，tf.divide(A,B):逐元素相除

tf.pow(A,B):逐元素求幂，tf.square(A):逐元素求平方

tf.sqrt(A):逐元素开平方，tf.exp(A):逐元素计算e的A次方

tf.one\_hot(indices,depth):创建深度为depth的独热编码，独热编码使取值不具有偏序性

tf.where(array<a,b,c):array中元素小于a返回b，否则返回c

tf.square(A)：求A中每个元素的平方

tf.argmax(A,axis),axis=0:求每列最大值的索引(axis默认为零）

tf.reduce\_mean(A,axis)：axis=0求每列的均值（不指定求所有的均值）

tf.reduce\_sum(A,axis)：axis=0求每列的和（不指定求所有的和）

tf.reduce\_max(A,axis)：axis=0求每列的（不指定求所有的最大值）

with GradientTape as GT://loss=...//grads=GT.gradient(loss,[w,b]):求w,b的损失值

**tf.keras：Tensorflow的标准高阶API，keras也有单独用的库**

keras:开源人工神经网络库，安装tensorflow时自动安装kera

compose(A,B,C):依次执行A,B,C4层，前一个输出作为后一个输入

**方式1：**model=tf.keras.Sequential()

/ model.add(layer1)：添加层，但Sequential不能构建复杂的结构

或者：model=tf.keras.Sequential(layer1,layer2...)

tf.keras.Sequential(\*(layer1 for i in range(n)))：构建相同层

**方式2：**input\_shape=tf.keras.layers.Input(...)

/ x=tf.keras.layers...(...)(input\_shape)

/ x=tf.keras.layers...(...)(x)：不断叠加层，第一层必须为输入层。

/ model=tf.keras.Model(input\_shape,[x1,x2,...],name=’’):组合

**方式3：**class model(tf.keras.layers.Layer) / def \_\_init\_\_(self,...) / super().\_\_init\_\_() / self.layer1=layer1...：初始化A=model(\_\_init\_\_中参数)时会搭建好模型结构，因此\_\_init\_\_中要把所有用到的层按顺序传入self(tf.keras.layers.Layer为搭建模型的继承类库函数)

/ def call(self,input1) / return x（）：调用A(input1)会直接执行call函数，此时才会确定模型内部的连接方式。对于复杂的网络定义多个类模块嵌套，类中可以使用Sequential函数

**基础网络：**

tf.keras.layers.Flatten()：将多维数据变成一维。输入(batch,...)，输出(batch,n)

tf.keras.layers.Dense(a,activation,name)：全连接层，输入(batch,in\_features)，输出(batch,out\_features)。activation有’relu’’softmax’’sigmoid’等；input\_shape=(b,)中b是输入一维数据个数

tf.keras.layers.Dropout(0.2,name)：丢弃层，输入/输出(batch,n)

**二维网络：**

tf.keras.layers.Input(shape=(a,a,b),name))：指定输入形状

tf.keras.layers.Conv2D(filters,kernel\_size,strides=1,padding=’valid’,activation=None,name)：二维卷积层。filters是卷积核数；kernel\_size=(n,n)/n是卷积核大小；strides为步长；padding是扩充边界方式，’same’为不够时扩充0（步长为1时使输入和输出一致，优先扩充右和下）、’valid’不够时舍弃；activation是激活函数(可不加):softmax,sigmoid,tanh,relu,softplus等经典激活函数

tf.keras.layers.BatchNormalization(name)：批归一化层

tf.keras.layers.Activation(...)：经典激活函数。mish函数可通过叠加softplus和tanh激活函数实现

tf.keras.layers.LeakyReLU(alpha=0.05)：LeakyReLU激活函数

tf.keras.layers.MaxPool2D(pool\_size,strides,padding,name)：二维最大池化层。pool\_size=(n,n)，strides为步长

tf.keras.layers.AveragePooling2D（pool\_size,strides,padding,name)：二维平均池化层

tf.keras.layers.concatenate([A,B],axis=3/-1,name)：合并分支

def inception1(x,...)//branch1=...(...)(x)//branch2=...(...)(x)//branch2=...(...)(branch2)//x=...concatenate(...)//return x

input1=...Input(...)//x=...(...)(input1)//x=inception1(x,...)：把分支加入网络

tf.keras.layers.add([A,B],name)：将两个完全一样的层参数相加

tf.keras.layers.UpSampling2D(size=a)：复制上采样层。扩大为a倍

tf.keras.layers.Conv2DTranspose(filters,kernel\_size,strides=1,padding=’valid’,name)：反卷积上采样层。stride决定放大的倍数；output\_padding为输出后每条边填充0

**二维损失：**

tf.keras.losses.categorical\_crossentropy(A,B):求A,B每行的交叉熵

**一维网络：**

tf.keras.layers.LSTM()：创建LSTM模块

tf.keras.layers.Conv1D(filters,kernel\_size,strides,padding,activation,name)：卷积层

**数据处理：**

**设置：**

tf.random.set\_seed(a)：设置全局随机种子a，使实验结果一样

**模型训练：**

model.summary():查看模型摘要。model.compile(loss,optimizer,[metrics1,metrics2]):配置模型；loss有均方差：'mse'，多分类交叉熵：'categorical\_crossentropy'（标签独热编码），'sparse\_categorical\_crossentropy'（标签自然顺序码），'binary\_crossentropy'（二分类交叉熵）；optimizer有'sgd''adagrad'，'adadelta'，'adam'；metrics有'accuracy'（标签数值，预测数值）'binary\_accuracy'（二分类，标签数值，预测概率），'categorical\_accuracy'（多分类，标签独热编码，预测独热编码），'sparse\_categorical\_accuracy’（多分类，标签数值，预测独热编码）

model.fit(train\_x,train\_y,batch\_size,epochs,shuffle,validation\_split,validation\_freq,verbose):训练模型，可反复运行。batch\_size是把总数据分成几份，一般取；epoch是训练次数；shuffle默认True是每次训练前打乱数据；validation\_split=0-1是分配测试集百分比默认为0；validation\_freq是训练几轮显示一次默认为1；verbose是显示方式：0为不显示，默认为1方式显示。

model.history.history:查看训练时记录的各指标。model.history.history[‘loss’]得到损失函数历史列表

model.evaluate(test\_x,test\_y,,batch\_size,verbose):用测试集评估

**模型保存与加载：**

model.predict(x,batch\_size,verbose):使用模型

model.load\_weights(path.h5)：读取网络参数

model.save\_weight(path.h5)：保存网络参数

model.save(filepath,overwrite,include\_optimizer,save\_format):保存模型；filepath是保存路径默认当前路径；overwrite默认为Ture表示保存的文件名相同时覆盖原文件；include\_optimizer默认为True表示保存优化器当前状态(训练时间过长时，下次可以接上断点继续训练)；save\_format默认为None,None保存为HDF5格式，文件后缀.h5，‘tf’保存为Tensorflow特有格式，会出现多个子文件

tf.saved\_model.load(model\_file):加载保存的模型文件。model\_file为一个特殊格式的文件夹

**import torch**

pytorch官网：https://pytorch.org/get-started/previous-versions/

(pip install torch：cpu版本，GPU版本详见官网，需要有对应的cuda)

torch中tensor是并行计算。默认float32和int64，CPU上支持float32，GPU支持float32和float16(部分GPU只支持float32，用float16不报错，但实际是转化为float32计算)。tensor中的运算要比np快

.cuda()/.cpu()/.to(‘cpu’/‘cuda’/’cuda:0’/torch.device(...),non\_blocking=False)：将数据/模型放到GPU/CPU上计算，’cuda:0’代表放入的GPU设备号，单卡时cuda和cuda:0等效；non\_blocking为是否放到锁存上(主机内存够时放到锁存上会快一点，但第一次放入gpu锁存会比较慢，因此如果预测程序是断开的不要放)。np数组只能在CPU上运算，张量能在CPU/GPU上运算

torch中tensor1=tensor2是共用内存的，会共同改变

**基本运算：大部分和numpy一致**

tensor1=tensor2.clone()：复制张量。torch中张量直接用=是共用内存的

torch.tensor(array,dtype=torch.float32,device=’cpu’)：创建torch.tensor张量，value可为列表、numpy数组等，建议将对象转为np再转为张量更快速。device=’cuda’为在GPU上创建tensor

torch.rand(a,b,dtype)：创建形状为(a,b)、0-1均匀分布的随机张量

torch.randperm(a)：相当于torch.arrange再打乱顺序

torch.from\_numpy(array)：将数组转化为torch.tensor张量

tensor.shape：张量形状

tensor.device：张量所在的设备

tensor.type(torch.int32/float32)：转换数据类型为int32/float32

tensor.tolist()：转换为列表

tensor[mask]：mask为布尔张量，可以筛选出mask中为True的值

tensor.numpy():将tensor张量变为numpy数组。当tensor放到模型中计算后，转为numpy需要阻断反向传播：tensor.detach().cpu().numpy()

tensor.to(torch.float32)：数据类型转换

torch.concat/cat([tensor1,...],dim=0)：不增加维度的方式合并张量

torch.stack([tensor1,tensor2...],dim=0)：增加维度的方式合并张量

torch.clamp(input,min,max)：将input中的值压缩到min到max之间

tensor.sigmoid()：进行sigmoid函数运算

torch.sort(array,axis=0,descending=False)：根据axis维度的值排序。descending=True时为降序排序

torch.argsort(array,axis=0,descending)：根据axis维度的值排序，返回下标

**维度转换：**

torch.reshape(tensor,(a,b))：变换张量形状。为(-1,)时变一维

tensor.permute(0,2,1)：更改维度，0,2,1表示将原来的1,2维换位

tensor.unsqueeze(a)：在a维添加一个维度

tensor.squeeze(a)：当a维度大小为1时去掉

**torch.nn：搭建神经网络**

对于网络需要输入形状时，输入形状后系统会把batch理解为批量，然后并行计算。网络搭建时最好不用if判断选择网络结构，后续转为onnx模型时容易出错

**方式1：**model=torch.nn.Sequential()

//model.add\_module(‘name’,layer1)：添加层。但不能构建复杂结构

或者：model=torch.nn.Sequential(layer1,layer2...)：名字自动生成

torch.nn.Sequential(\*(layer1 for i in range(n)))：构建相同层

**方式2：**class model(torch.nn.Module) // def \_\_init\_\_(self,...) // super().\_\_init\_\_() // self.layer1=layer1...：初始化A=model(\_\_init\_\_中参数)时会搭建好模型结构，因此\_\_init\_\_中要把所有用到的层按顺序传入self(torch.nn.Module为搭建模型的继承类库函数)

//def forward(self,x) // return x：调用A(x)会直接执行forward函数，此时才会确定模型内部的连接方式。对于复杂的网络定义多个类模块嵌套，类中可以使用Sequential函数。forward中用if可能导致onnx导出失败

with torch.no\_grad()：停止自动求导。自定义loss时和验证时使用(官方loss自带)

**基础网络：**

torch.nn.Sigmoid/Tanh/ReLU/LeakyReLu/...()：添加激活函数

torch.nn.Flatten()：将多维数据变成一维。输入(batch,...)，输出(batch,n)

torch.nn.Linear(in\_features,out\_features)：全连接层。输入(batch,in\_features)，输出(batch,out\_features)。多变量预测时可(batch,dim,in\_features)，输出(batch,dim,out\_features)

torch.nn.Dropout(0.2)：丢弃层，输入/输出(batch,n)

torch.nn.Softmax(dim)：将输出结果归一化。dim是进行归一化的维度。如果是单类别不能用softmax而要用sigmoid。softmax函数之后再加网络输出层可能会输错

**一维网络：**

torch.nn.Conv1d(in\_,out\_,kernel\_size,stride=1,padding=0,dilation=1,bias=True)：卷积层。输入(batch,dim,n)，in1/ou1为输入/输出维度，对应dim；stride和padding对应n，stride为卷积步长(卷积后四周会缩短stride//2)；padding=a在序列首尾填充a个0；dilation为卷积时卷积核各点的距离(形成空洞卷积)；bias=True在输出添加可学习偏差

output,(h\_n,c\_n)=torch.nn.LSTM(input\_size,hidden\_size,num\_layers,dropout=,bidirectional=False)：创建LSTM模块。input\_size为输入的长度；hidden\_size为输出的长度；num\_layers为层数数；dropout为在除最后一层外都加丢弃层；bidirectional为是否双向传播。LSTM加在卷积后效果很差

torch.nn.BatchNorm1d(in1)：批归一化层。in1为输入的维度

torch.nn.MaxPool1d(kernel\_size,stride,padding)：最大池化层

torch.nn.Upsample(scale\_factor=a)：上采样层。a为扩大多少倍

**一维损失：**

torch.nn.L1Loss()(a,b)：平均误差。预测值和真实值绝对误差和的平均

torch.nn.MSELoss()(a,b)：均方误差。预测值和真实值误差平方和的平均

**二维网络：**

torch.nn.Conv2d(in\_,out\_,kernel\_size,stride=1,padding=0,dilation=1,bias=True)：卷积层。输入(batch,dim,m,n)，in1/ou1为输入/输出维度，对应dim；stride和padding对应m、n，stride为卷积步长(卷积后四周会缩短stride//2)；padding=a在图片周围填充a个0；dilation为卷积时卷积核各点的距离(形成空洞卷积)；bias=True在输出添加可学习偏差

torch.nn.BatchNorm2d(in1,eps=1e-05,momentum=0.1)：批归一化层。in1为输入的维度；eps为防止分母为零的小值，推荐0.001；momentum为均值和方差的额估计，推荐0.03

torch.nn.MaxPool2d(kernel\_size,stride,padding,dilation=1)：最大池化层。stride默认为kernel\_size的值，dilation>1为空洞卷积

torch.nn.Upsample(scale\_factor=a)：复制上采样层。扩大为a倍

torch.nn.ConvTranspose2d(in1,out1,kernel\_size,stride=1,padding=0,output\_padding=0)：反卷积上采样层。stride决定放大的倍数；output\_padding为输出后每条边填充0

torch.nn.AdaptiveAvgPool2d(n)：将尺寸进行平均变为1。如n=1时输入(batch,16,20,20)，输出(batch,16,1,1)

**二维损失：**

torch.nn.BCELoss(weight=None,reduction=‘mean’)(pred,true)：交叉熵损失(Binary Cross Entropy)。Pred,true为0-1的值，独热编码。输入的weight为权重，要和标签形状一致；reduction：’mean’返回损失的平均值。使用BCELoss时无法使用混合精度计算

torch.nn.torch(...)：先进行sigmoid归一化再进行交叉熵，推荐使用。使用混合精度计算时要用BCEWithLogitsLoss

**数据处理：**

先把数据读取存放在CPU中，训练时再用Dataset-Dataloader的方法取批量数据到GPU中。对于大量的数据集，使用Dataset-Dataloader不用预先把数据转换为张量存放在GPU中。

class dataset(torch.utils.data.Dataset) / def \_\_init\_\_(self,参数) //def \_\_len\_\_(self) //return len(训练数据长度) //def \_\_getitem\_\_ (self,index) //... //return image,label：通过继承定义数据集类。\_\_len\_\_为数据集的长度。\_\_getitem\_\_中写入数据读取和处理的函数，返回结果，index为系统参数，会自动在\_\_len\_\_长度内选取一个索引。在\_\_getitem\_\_中只对单个的数据处理，实际调用时DataLoader函数会用多个进程在模型训练时提前并行处理多个数据，最终合成一个批量

dataloader=torch.utils.data.DataLoader(dataset(参数),batch\_size,shuffle,drop\_last,num\_workers,pin\_memory=False,sampler=None,collate\_fn)：根据Dataset批量划分训练数据。batch\_size为批量大小；shuffle=True每次取数据前打乱数据；drop\_last=True时如果最后一批数据不满足批量时舍弃，False时把剩余数据当作一个批量；num\_workers为有几个进程处理数据，0为所有数据加载到主进程，可以开多个以保证数据预处理的效率，预测时一般只用一个；pin\_memory=True把数据放到CPU锁存上，之后再放到GPU上的时间会加快（此时\_\_getitem\_\_中不能先把数据放到GPU上）；sampler为取数据的方式，一般只在分布式训练时设置sampler=torch.utils.data.distributed.DistributedSampler(dataset)，同时设置shuffle为False，否则各GPU会运行相同的dataloader，设置sampler会让各个GPU取的index错开；collate\_fn为取数据的方式，默认将每个\_\_getitem\_\_返回的结果合在一起并转为张量(增加在0维上)

//自定义collate\_fn：def collate\_fn(batch): // return train\_batch,true\_batch。当执行完batch个\_\_getitem\_\_后，所有\_\_getitem\_\_返回的值作为列表会传入到batch中，经过处理后变成最终的train\_batch,true\_batch。使用方式为collate\_fn=collate\_fn

//使用多个num\_workers时，相当于把dataloader分为num\_workers份，每个线程中在每轮训练时都会重新进行self的初始化

for item,(train\_batch,true\_batch...) in enumerate(dataloader)：取出准备好的数据(自动转为张量),每次for循环都是新的一轮。加enumerate时前面一定要带()区分item

**设置；**

torch.cuda.is\_available()：判断cuda是否可用返回True或False

torch.manual\_seed(种子号)：为CPU设置随机种子

torch.cuda.manual\_seed(种子号)：为当前GPU设置随机种子

torch.cuda.manual\_seed\_all(种子号)：为所有GPU设置随机种子

torch.backends.cudnn.deterministic=True：固定每次返回的卷积算法

torch.backends.cudnn.enabled =True：cuDNN使用非确定性算法

torch.backends.cudnn.benchmark=True：在训练前cuDNN会先搜寻每个卷积层最合适实现它的卷积算法，加速运行；但对于复杂变化的输入数据，可能会有过长的搜寻时间，对于训练比较快的网络建议设为False

**加载和保存模型：**

torch.save(model/model\_dict,’path.pt’)：保存模型搭建信息(不是真正的模型)和模型参数。torch.save本质是让目标序列化，可以保存嵌套模型、学习率等信息的字典，此时torch.load加载后还需逐个取出

torch.load(’path.pt’,map\_location='cpu'/'gpu')：根据模型信息和代码搭建模型和传入参数。map\_location等同于.to(device)。加载模型需要保存模型时一摸一样的代码

torch.save(model.state\_dict(),’path.pt’)：保存模型参数

model.load\_state\_dict(torch.load(’path.pt’)：加载模型参数

**模型训练：**

.pkl/.pt/,pth：pytorch模型文件，三者只是文件后缀不同，是同一文件

torch.optim.SGD(model.parameters(),lr=0.001)：SGD优化器。model.parameters()为模型的网络参数；lr为学习率；weight\_decay

torch.optim.Adam(model.parameters(),lr=0.001,betas=(0.9,0.999),eps=1e-08,weight\_decay=0)：Adam优化器。model.parameters()为模型参数；lr为学习率；betas为计算梯度以及梯度平方的运行平均值的系数；eps为防止分母为0；weight\_decay为权重衰减。yolov7中betas=(0.937,0.999)，weight\_decay=0.0005

model.float()/.half()：转换模型计算使用的精度，要和输入张量一致。CPU只支持float32，GPU支持float32和float16

1，model.train().float()：加在训练前，启用批归一化层和丢弃层

2,optimizer=torch.optim.Adam(model.parameters(),lr)：优化器

3，pred=model(data)：将数据输入到模型中，正向传播，会自动求导

4，loss\_value=loss(pred,true)：根据输出计算损失。train\_loss.item()：损失的值。自定义loss中要加with torch.no\_grad()(官方loss自带)

5，optimizer.zero\_grad()：将参数的梯度设为0。网络参数进行反馈时梯度会积累，每轮梯度需要清零重新计算

6，loss\_value.backward()：计算当前梯度，反向传播。如果报错可能是损失函数/网络结构的问题

7，optimizer.step()：更新参数

**学习率保存和加载：继续训练模型**

optimizer=torch.optim.Adam(model.parameters(),lr)：优化器

model\_dict[‘optimizer\_state\_dict’]=optimizer.state\_dict()：记录优化器参数

optimizer.load\_state\_dict(model\_dict[‘optimizer\_state\_dict’])：加载优化器参数，但此时的优化器optimizer要和之前的优化器保持一致

**混合float16精度训练：**

使用混合精度时，很小的数值会被阶段为0，如果网络中有log(0)会导致输出全为nan，sigmoid等归一化需要注意。使用混合精度训练时，要使用BCEWithLogitsLoss，不能使用BCELoss。在windows上使用混合精度可能会出现nan，但在Linux上正常

scaler=torch.cuda.amp.GradScaler()：自动使用混合精度训练装置

with torch.cuda.amp.autocast()://模型推理//损失计算：使用混合精度训练

optimizer.zero\_grad()：梯度清理

scaler.scale(loss\_value).backward()：替代原来的loss\_value.backward()

scaler.step(optimizer)：替代原来的optimizer.step()

scaler.update()：装置本身也需要更新

**指数移动平均(EMA)：给予近期数据更高权重的平均方法**

注意：不能将ema放到args中，否则会导致模型保存出错

ema=ModelEMA(model)：ModelEMA为自定义函数(yolov7代码中有)

ema.updates=0：新建模型开始训练时要设置为0，每轮batch迭代使用ema.update时会自动+1，加载已有模型时要赋值为之前的数据

ema.update(model)：每个batch迭代后、损失更新参数后使用EMA调整参数

model=ema.ema：代替model.eval()进行验证，因为ema更新参数只计算训练时的，因此测试时要单独复制一个模型进行验证

model\_dict['ema\_updates'] = ema.updates：保存参数，方便下次训练

**分布式训练：只有多卡下能使用**

torch.nn.DataParallel：单机多卡数据并行训练(训练效率无法达到最高)

torch.nn.parallel.DistributedDataParallel：多机多卡数据并行训练(推荐)

对模型使用DataParallel/DistributedDataParallel后，模型会被附加一个状态，因此保存模型时要保存model.module

同一份代码会在不同GPU上运行，先把模型放到不同GPU上，每个GPU对一个batch推理和计算损失（先完成的GPU会一直等待），之后将损失结果平均并进行反向传播，所有GPU上的模型同步更新参数。验证和计算指标也可以设置为分布式。模型保存等代码只需在主设备上运行。使用local\_rank==0判断是否在主设备上

使用分布式时，计算的损失无法汇总，因此训练集的损失只记录主设备上的，验证集要计算所有损失需要

假设总数据为m，有n个GPU，dataloader中设置batch=a，dataset中写法不变，\_\_len\_\_依然为m，则实际的batch为，迭代次数为

torch.cuda.device\_count()：统计使用的GPU数

1，local\_rank=int(os.environ["LOCAL\_RANK"])：当前GPU在服务器的GPU排号

int(os.environ["RANK"])：当前GPU在所有服务器的GPU排号

2，torch.distributed.init\_process\_group(backend)：分布式训练初始化。backend为GPU上模型同步更新参数时通信使用的后端，一般为’ncll’

3，dataloader中设置sampler=torch.utils.data.distributed.DistributedSampler(dataset)，同时不设置shuffle：index的值是在sampler得到的，不设置sampler时默认所有GPU取到的index一样，设置sampler会让各个GPU取的index错开

3，model=model.to(‘cuda:n’)：将模型放到当前设备上，n为GPU排号

或者torch.cuda.set\_device(local\_rank)

4，torch.nn.parallel.DistributedDataParallel(model,device\_ids=[local\_rank], output\_device=local\_rank,find\_unused\_parameters=False))：对放到设备上的模型进行一些初始化。device\_ids是指定设备的id，output\_device是指定设备的位置，通常都是local\_rank；find\_unused\_parameters=True时会找出模型中没有梯度的参数，在更新参数时会加快速度，如果网络中有很多没有梯度的参数时应开启，否则反而会略微降低速度

5，进行训练之后，模型会自动平均所有GPU上的损失，所有模型同步进行参数更新。模型保存等代码只在local\_rank==0时运行

6，torch.distributed.barrier()：每轮训练结束后同步一下，让快的GPU等下慢的GPU，主进程的GPU要进行验证，更新参数时会自动同步不需要加。

7，在命令行传入参数来使用分布式多线程：python -m torch.distributed.launch --master\_port 9999 --nproc\_per\_node 2 文件名.py。master\_port为各GPU之前的通信端口(不被占用就行)，nproc\_per\_node为使用的GPU数。使用args时可以留有参数--local\_rank，执行命令后会自动分配每个GPU的local\_rank给每个GPU的代码

**清理显存空间：**

del ...：删除一些无用但还在显存上的变量

torch.cuda.empty\_cache()：释放一些没用的显存占用，有较明显的效果

**模型测试：**

torchvision.ops.nms(boxes,scores,iou\_threshold)：官方非极大值抑制，返回索引index组成的张量,使用boxes[tensor]取出结果。boxes为(x\_min,y\_min,x\_max\_y\_max)。torchvision为pytorch辅助库

1，model.eval().float()/.half()：转为测试时的精度(需要GPU且支持float16)

2，with torch.no\_grad()：测试模型时关闭反向求导

3，pred=model(data).detach().cpu()：阻断反向传播并放到cpu上

4，val\_loss=loss(pred,true)

**pytorch转onnx：**

模型转换时需要用输入数据运行一次，转出的onnx模型数据类型和pt模型一致，如果pt模型要转为float16，pt模型需要放到GPU上。pt模型搭建时尽量不要在模型forward中用if判断，容易导致onnx模型运行出错

1，model=torch.load(path,map\_location)：加载模型

(模型中合并卷积和归一化层以提高推理速度)

2，model.float()/.half().eval()：模型处理

3，torch.onnx.export(model,任一输入,保存位置,opset\_version,input\_names=[],output\_names=[],dynamic\_axes)：将pytorch模型转为onnx。x为任一输入如torch.randn(1,3,640,640).to(device)，用于确定输入形状和执行一次推理以确定模型连接方式，；path为保存路径；opset\_version为opset版本，尽量用高版本；input\_names为输入的名称，一般只有一个输入如[‘input’]；output\_names为输出的名称如['output']，如果模型带上非极大值抑制会有多个如['num\_dets','det\_boxes','det\_scores','det\_classes']；dynamic\_axes={'input':{0:'batch\_size'},'output':{0:'batch\_size'}}为指定输入输出(输出可能有多个)是否为动态，0为动态

4，简化onnx模型，可用onnxsim库

**模型部署：**

详见onnx和tensorrt库

**import torchvision：pytorch辅助库**

安装要与torch版本对应

**import timm：集成了很多pytorch的图像模型**

官方项目：<https://github.com/rwightman/pytorch-image-models>

运行后下载的模型文件位置在本地用户中

timm.list\_models(pretrained=False)：查看所有模型名称，返回列表。pretrained=True时只查看可传入训练参数的模型，大部分都可传入

timm.list\_models('\*resnet\*')：搜索名称中带有resnet的所有模型，返回列表，没有搜到为空列表

timm.create\_model('模型名',pretrained=False,features\_only=False,in\_chans,num\_classes,exportable)：创建模型。pretrained=True时会下载模型预训练权重参数到本地，可以加快训练速度；features\_only为True时只提取特征；in\_chans指定输入维度，num\_classes指定输出类别数(features\_only=False才生效)，模型输出没有经过归一化；exportable=True方便导出onnx

**import transformers：集成了很多pytorch的自然语言处理模型**

官方项目：<https://github.com/huggingface/transformers>

模型：<https://huggingface.co/models>

运行后下载的模型文件位置在本地用户中

tokenizer=transformers.AutoTokenizer.from\_pretrained(官方模型名称/本地模型文件夹位置)：加载模型的预处理方式

tokenizer.add\_tokens([词语])：添加编码组合。比如默认每个字编码单独编码，加入词后会将整个词一起编码

dict=tokenizer([句子1,句子2...],padding=False,truncation,max\_length,return\_tensors)：对数据进行预处理，返回字典{'input\_ids','token\_type\_ids','attention\_mask'}，input\_ids会对每个字符编码并在前后加上开始、结束编码。padding=True在后面填充0直到与最长序列相等；truncation=True截断为max\_length长度，max\_length小于最长序列时才生效；return\_tensors="pt"使返回的结果为torch的张量

tokenizer.decode(dict[‘input\_ids’][i])：解码为句子(一次只能解码一句)，会带上[CLS]、[SEP]标识

tokenizer.add\_tokens([多个字符,...])：正常情况下会将一个字符编码为一个数字，使用后会将添加的多个字符一起编码

model=transformers.AutoModel.from\_pretrained(官方模型名称/本地模型文件夹位置).to(device)：加载官方模型/本地模型位置

model=transformers.AutoModelForSequenceClassification.from\_pretrained(官方模型名称/本地模型文件夹位置).to(device)：加载模型

model=transformers.AutoModelForTokenClassification.from\_pretrained(官方模型名称/本地模型文件夹位置).to(device)：加载序列标注模型

model=transformers.AutoModelForQuestionAnswering.from\_pretrained(官方模型名称/本地模型文件夹位置).to(device)：加载问答模型

model=transformers.AutoModelForSeq2SeqLM.from\_pretrained(官方模型名称/本地模型文件夹位置).to(device)：加载翻译模型

model=transformers.pipeline(‘任务类型’,model=model,tokenizer=tokenizer,device,...)：将编解码与模型组合到一起，直接传入句子，每个模型会有自己的自定义参数。

pred=model(input\_batch)：使用模型预测

**nllb：facebook大型语言翻译模型，能够在200种语言之间翻译**

官方项目：<https://github.com/facebookresearch/fairseq/tree/nllb/examples/nllb/modeling>

tokenizer=transformers.AutoTokenizer.from\_pretrained(r"D:\model\nllb-200-distilled-600M")

model=transformers.AutoModelForSeq2SeqLM.from\_pretrained(r"D:\model\nllb-200-distilled-600M")

model=transformers.pipeline("translation",tokenizer=tokenizer,model=model,truncation, max\_length,src\_lang,tgt\_lang)：加载nllb模型。src\_lang为输入语言，tgt\_lang为输出语言，zho\_Hans为中文简体，zho\_Hant为中文繁体，eng\_Latn为英语

**import paddlepaddle：paddle框架**

官方项目：<https://github.com/PaddlePaddle/Paddle>

API:https://www.paddlepaddle.org.cn/documentation/docs/zh/api/index\_cn.html

**基本操作：大部分和numpy一致**

paddle.to\_tensor(A)：转换为张量

**paddle.nn：搭建神经网络（和pytorch大体一致）**

对于网络需要输入形状时，输入形状后系统会把batch理解为批量，然后并行计算。网络搭建时最好不用if判断选择网络结构，后续转为onnx模型时容易出错

**方式1：**model=paddle.nn.Sequential()

//model.add\_sublayer(‘name’,layer1)：添加层。但不能构建复杂结构

或者：model=paddle.nn.Sequential(layer1,layer2...)：名字自动生成

paddle.nn.Sequential(\*(layer1 for i in range(n)))：构建相同层

**方式2：**class model(paddle.nn.Layer) // def \_\_init\_\_(self,...) // super().\_\_init\_\_() // self.layer1=layer1...：初始化A=model(\_\_init\_\_中参数)时会搭建好模型结构，因此\_\_init\_\_中要把所有用到的层按顺序传入self(paddle.nn.Layer为搭建模型的继承类库函数)

//def forward(self,x) // return x：调用A(x)会直接执行forward函数，此时才会确定模型内部的连接方式。对于复杂的网络定义多个类模块嵌套，类中可以使用Sequential函数。forward中用if可能导致onnx导出失败

**基础网络：**

paddle.nn.Sigmoid/Tanh/ReLU/LeakyReLu/...()：添加激活函数

paddle.nn.Flatten()：将多维数据变成一维。输入(batch,...)，输出(batch,n)

paddle.nn.Linear(in\_features,out\_features)：全连接层。输入(batch,in\_features)，输出(batch,out\_features)。多变量预测时可(batch,dim,in\_features)，输出(batch,dim,out\_features)

paddle.nn.Dropout(0.2)：丢弃层，输入/输出(batch,n)

**一维网络：**

paddle.nn.Conv1D(in\_,out\_,kernel\_size,stride=1,padding=0,dilation=1)：卷积层。输入(batch,dim,n)，in1/ou1为输入/输出维度，对应dim；stride和padding对应n，stride为卷积步长(卷积后四周会缩短stride//2)；padding=a在序列首尾填充a个0；dilation为卷积时卷积核各点的距离(形成空洞卷积)

**二维网络：**

paddle.nn.Conv2D(in\_,out\_,kernel\_size,stride=1,padding=0,dilation=1)：卷积层。输入(batch,dim,m,n)，in1/ou1为输入/输出维度，对应dim；stride和padding对应m、n，stride为卷积步长(卷积后四周会缩短stride//2)；padding=a在图片周围填充a个0；dilation为卷积时卷积核各点的距离(形成空洞卷积)

paddle.nn.AdaptiveAvgPool1D(output\_size)：平均池化。output\_size=1时，输入(batch,dim,n)，输出(batch,dim,1)

**import paddleslim：飞桨模型量化库**

**import paddlenlp：飞桨自然语言处理库**

官方项目：<https://github.com/PaddlePaddle/PaddleNLP>

API：<https://paddlenlp.readthedocs.io/zh/latest/>

使用paddle模型，需要先安装paddlepaddle库

tokenizer=paddlenlp.transformers.AutoTokenizer.from\_pretrained(官方模型名称/本地模型文件夹位置)：加载模型的预处理方式

tokenizer.add\_tokens([词语])：添加编码组合。比如默认每个字编码单独编码，加入词后会将整个词一起编码

dict=tokenizer([句子1,句子2...],padding=False,max\_seq\_len,truncation=False,max\_length)：对数据进行预处理，返回字典{'input\_ids','token\_type\_ids'}，input\_ids会对每个字符编码并在前后加上开始、结束编码，没有attention\_mask，因为模型不需要提供。padding=True在后面填充0直到与最长序列相等；truncation=True截断为max\_length长度，max\_length小于最长序列时才生效

input\_ids=paddle.to\_tensor(dict[’input\_ids’])：转为张量

tokenizer.decode(input\_ids[i])：解码为句子(一次只能解码一句)，会带上[CLS]、[SEP]标识

tokenizer.add\_tokens([多个字符,...])：正常情况下会将一个字符编码为一个数字，使用后会将添加的多个字符一起编码

model=paddlenlp.transformers.AutoModel.from\_pretrained(官方模型名称/本地模型文件夹位置)：加载模型

model=paddlenlp.transformers.AutoModelForSequenceClassification.from\_pretrained(官方模型名称/本地模型文件夹位置,num\_classes)：加载文本分类模型。num\_classes为句子的类别数

model=paddlenlp.transformers.AutoModelForTokenClassification.from\_pretrained(官方模型名称/本地模型文件夹位置)：加载序列标注模型。num\_classes为句子中每个字符的类别数

model=paddlenlp.transformers.AutoModelForQuestionAnswering.from\_pretrained(官方模型名称/本地模型文件夹位置)：加载问答模型

**ernie-3.0：百度提出的超大语言模型，在多个数据集上均优于以往模型**

官方项目：<https://github.com/PaddlePaddle/PaddleNLP/tree/develop/model_zoo>

paddlenlp.transformers.AutoModel.from\_pretrained('ernie-3.0-medium-zh')：加载ernie模型

**import fastdeploy：飞桨模型部署库**

官方项目：<https://github.com/PaddlePaddle/FastDeploy>

函数：<https://baidu-paddle.github.io/fastdeploy-api/python/html/>

(pip install fastdeploy-python/fastdeploy-gpu-python -f <https://www.paddlepaddle.org.cn/whl/fastdeploy.html)>

(cpu的只支持CPU推理，gpu的支持CPU和GPU，tensorrt只能在GPU上)

支持padding和pytorch模型，需要先安装cuda和cudnn，不需要安装tensorrt

飞桨框架的模型分为模型结构和参数两部分

runtime\_option=fastdeploy.RuntimeOption()：后端推理配置

runtime\_option.use\_cpu()/.use\_gpu()：使用CPU/GPU

runtime\_option.use\_ort\_backend()：Paddle/ONNX模型用ort框架推理

runtime\_option.use\_trt\_backend()：Paddle/ONNX用trt框架GPU推理

runtime\_option.set\_trt\_input\_shape(‘输入名称’,形状如:[1, 3, 640, 640])：设置输入形状，有的模型还需要设置动态形状范围

runtime\_option.enable\_trt\_fp16()：trt推理时使用float16(默认32)

runtime\_option.set\_trt\_cache\_file(缓存位置.trt)：使用trt推理时会先转换为trt模型，如果存在缓存位置则直接加载，不存在则下次转换后会保存

**YOLOV7目标检测：**

runtime\_option.set\_trt\_input\_shape("images", [1, 3, 640, 640])

fastdeploy.vision.detection.YOLOv7(model,params\_file=None,runtime\_option=None,model\_format=fastdeploy.ModelFormat.ONNX)：构建YOLOV7推理模型。model可为.onnx/.pdmodel；params\_file当模型为onnx时无需设置；runtime\_option为后端推理配置；model\_format为模型格式，paddle为fastdeploy.ModelFormat.PADDLE

model.predict(image,conf\_threshold=0.25,nms\_iou\_threshold=0.5)：模型预测。image为WHC或BGR格式；conf\_threshold、nms\_iou\_threshold为阈值，如果导出为onnx模型时已经设置，因此只有此处阈值大于之前的才生效

**PPOCRv3文字识别：**

由目标检测、文字方向检测、文字识别三个模型组成。可直接从官网下载：

<https://github.com/PaddlePaddle/FastDeploy/tree/develop/examples/vision/ocr/PP-OCRv3/python>

det\_option.set\_trt\_input\_shape("x",[1,3,64,64],[1,3,640,640],[1,3,960,960])：推荐输入范围设置

det\_model=fastdeploy.vision.ocr.DBDetector()：构建目标检测模型

cls\_option.set\_trt\_input\_shape("x",[1,3,48,10],[cls\_batch\_size,3,48,320],[cls\_batch\_size,3,48,1024])：推荐输入范围设置

cls\_model=fastdeploy.vision.ocr.Classifier()：构建文字方向检测模型

rec\_option.set\_trt\_input\_shape("x",[1,3,48,10],[rec\_batch\_size,3,48,320],[rec\_batch\_size,3,48,2304])：推荐输入范围设置

rec\_model=fastdeploy.vision.ocr.Recognizer()：构建文字识别模型

model=fastdeploy.vision.ocr.PPOCRv3(det\_model=det\_model,cls\_model=cls\_model,rec\_model=rec\_model)：将三个模型串联组合起来

model.predict(image)

**import psutil：系统使用情况分析**

psutil.cpu\_times()：获取CPU情况

psutil.cpu\_count()：获取CPU逻辑个数

psutil.cpu\_count(logical=False)：获取CPU物理个数

psutil.virtual\_memory()：获取内存的使用情况

**import git：类似命令终端的git命令**

(pip install gitpython)

**labelimg：**标注图片保存标签为xml格式

终端输入labelimg打开软件

功能open dir：打开存放图片的文件夹。每次都要重新新设置

功能change save dir：设定标注存放的地址。A、D快速切换图片

功能view>>Auto Save mode：自动保存标注 | CTRL+S保存图片

按住CTRL加鼠标滚轮放大缩小

Create RectBox画框 | Delete删除框 | 可以选择拖动每个框

**import cv2**

(pip install opencv-python安装)

部分函数申请了专利要等保护器过后才能使用

cv2中读取图片、运算使用的是np数组组成的BGR通道(h,w,3)。cv2的HSV通道中H通道0-360简化为0-180。cv2中数据类型为np.uint8

cv2.\_\_version\_\_:查看版本；cv2.\_\_path\_\_:查看位置

opencv的读取RGB图片时，读取的顺序为BGR，转化的numpy数组类型为uint8，形状为(高,宽,通道)，内存要连续存储，对图像的操作通常会改变原图

np.ascontiguousarray(image)：将数组的内存变为连续存储

cv2.imread(路径,flags):读取图片转化为BGR的numpy数组(英文路径)，读取顺序为BGR。flags：默认为cv2.IMREAD\_COLOR忽略alpha通道；cv2.IMREAD\_GRAYSCALE/0读入灰度图；cv2.IMREAD\_UNCHANGED包括alpha通道

cv2.cvtColor(image,flag)：通道转换。flag：cv2.COLOR\_BGR2GRAY将BGR转为gray；cv2.COLOR\_BGR2RGB将BGR转为RGB；cv2.COLOR\_BGR2HSV将BGR转为HSV

capture=cv2.VideoCapture(路径):读取视频文件。路径为0时打开电脑摄像头，按S键保存图片

capture.isOpened():判断是否能打开，返回True或False

bool,image=capture.read():按顺序读取一帧(如果是摄像头则实时获取一帧)，返回bool值和图片

cv2.imwrite(路径,image):保存图片到指定位置，路径中包括名称。如果保存出的图片是黑色的，可能是因为image归一化的原因，此时需要image\*255

**显示图片：**

cv2.imshow(名称,image):将numpy数组显示成图片，显示通道为BGR

cv2.waitKey(0):0为一直显示(按任意键退出)，可改为任意的时长(毫秒)

cv2.destroyAllWindows():关闭所有窗口并释放内存(最好加上)

cv2.namedWindow(名称)：创建一个一直存在的窗口，然后使用cv2.imshow(名称要一致，否则会单独创建一个显示)和cv2.waitKey显示图片，最后再用cv2.destroyAllWindows()

**基本操作：**

image2=image1：引用image1 | image2=image1.copy()：复制image1

image[a:b,c:d]:选取图像的某个区域；image[:,:,0]:选取B通道

B,G,R=cv2.split(image1):取出每个颜色通道

image=cv2.merge((B,G,R)):通道合成

cv2.addWeighted(image,a,img2,b,c):融合图。a,b为融合比例，c为0

thresh,image2=cv2.threshold(image1,thresh,maxVal,type):转化为二值图,返回阈值和图像两个值。thresh为阈值，maxVal执行的操作一般为255，type：cv2.THRESH\_BINARY低于阈值设为0，高于阈值为maxVal，cv2.THRESH\_BINARY\_INV与之相反

cv2.inRange(image,a,b):将image中a到b的值变为255，其余变为0，当image为BGR时，a和b也为BGR三通到。把图像直接进行inRange操作的效果比转化为灰度图再做好

cv2.resize(image,(w,h),interpolation):将图像拉伸为指定形状(w,h)(注意与一般的reshape是反着的)。interpolation：cv2.INTER\_NEAREST最近邻插值，cv2.INTER\_LINEAR双线性插值（默认，速度快效果较好），cv2.INTER\_AREA使用像素区域关系重采样，cv2.INTER\_CUBIC为像素邻域的双3次插值，cv2.INTER\_LANCZOS4为像素邻域的Lancaos插值

cv2.copyMakeBorder(image,a,b,c,d,borderType):扩充边界。a,b,c,d为上下左右边界的扩充宽度，borderType为填充类型：cv2.BORDER\_REPLICATE复制边缘像素，cv2.BORDER\_REFLECT反射镜象，cv2.BORDER\_REFLECT\_101以边缘为轴反射镜像，cv2.BORDER\_CONSTANT,value=a填充常数a

cv2.flip(image,a)：翻转图像。a：1为水平翻转，0为垂直翻转，-1为水平垂直翻转

cv2.dnn.blobFromImage(image,size=(a,a),swapRB,mean,scalefactor=1)：将图片如(640,640,3)转为如(1,3,640,640)的形状并做预处理，数据类型也转为方便训练的float32。a为转换后图片的形状；mean=[r,g,b]为每个通道减去的均值，如果图片是BGR的，swapRB=True，不指定mean时不减均值；scalefacteor一般为1/255做归一化(减均值后)

cv2.dnn.blobFromImages([image,...])：处理多张图片时更快

**数据转换：**

cv2.imencode(‘.jpg’,image)：将图片用’.jpg’编码方式转为一行np.array数组

cv2.imdecode(np.array,cv2.IMREAD\_COLOR)：将一行np数组转换为BGR图片数组

**画图（会改变原图）：**数据不能为张量

cv2.line(image,point1,point2,color)：在image中画直线。point1/2为直线起始/终点坐标；color单通道为a，BGR通道为(a,b,c)

cv2.rectangle(image,point1,point2,color=(0,255,0),thickness=2)：在image中画矩形框。point1、point2为矩形左上、右下坐标(x,y)，且必须为整数型；thickness为框的宽度，必须为整数

cv2.circle(image,center,radius,color)：在image中画圆。center为圆心坐标，radius为半径长度

cv2.putText(image,’名称’,(x1,y1),font,a,(0,255,0),b)：画标签注释。x1,y1为标签左下坐标；font为字体如cv2.FONT\_HERSHEY\_SIMPLEX；a为字大小，一般为1以内；(0,255,0)为颜色；b为字粗细，必须为整数一般为1/2

cv2.fillConvexPoly(image,points,(a,b,c))：在image上根据points中的坐标画实心图案，(a,b,c)为颜色。points要为np.array

**形态学运算：**

cv2.bitwise\_and/or/add/subtract(image1,image2):与/或/加/减运算。越界时值为边界值

cv2.erode(image,kernel,iterations=):腐蚀，凹的地方变钝，凸的地方变尖。kernel为全1的卷积核：kernel=np.ones((a,a),np.uint8)；iterations=b为迭代次数

cv2.dilate(image.kernel,iteration):膨胀，凹的地方变尖，凸的地方变钝。kernel同上

cv2.morphologyEx(image,op,kernel):利用腐蚀和膨胀运算。kernel同上；op：cv2.MORPH\_OPEN开运算，先腐蚀再膨胀消除黑点；cv2.MORPH\_CLOSE闭运算，先膨胀再腐蚀消除小孔；cv2.MORPH\_TOPHAT顶帽，原图-开运算，突出亮的部分；cv2.MORPH\_BLACKHAT黑帽，闭运算-原图，突出暗的部分

cv2.calcHist([image],[channels],mask,[histSize],[ranges]):计算图像直方图，返回存放含histSize个元素的一维数组。channels为图像的通道；mask：一般为None处理全图；histSize表示使用多少柱子一般为256；ranges是像素的范围一般为[0,256]

cv2.equalizeHist(image):将image(单通道)直方图均衡化，返回图像

clahe1=cv2.createCLAHE(clipLimit=a,tileGridSize=(b,b)):定义一个clahe模板。clipLimit可为2，tileGridSize=(b,b)为划分的区域数量，分别对各个区域进行直方图均衡化，然后把区域边界额外处理后合成

clahe1.apply(image):将clahe1模型应用到image中，会改变image

**卷积运算与边缘检测：**

cv2.filter2D(image,ddepth,kernel):对图像进行卷积。ddepth：-1表示输出与原图像深度相同

cv2.blur(image,(a,a)):均值滤波。(a,a)为卷积核，一般卷积核较大

cv2.GaussianBlur(image,(a,a),sigmaX,sigmaY):高斯滤波，比均值效果好。(a,a)为卷积核，sigmaX/Y为x和y轴上的高斯核标准偏差，偏差越大越模糊

cv2.medianBlur(image,a):中值滤波，适合去除噪音点。a为方框大小

cv2.Sobel(image,ddepth,dx,dy,ksize):用Sobel算子计算，会有负数。ddepth一般为cv2.CV\_64F；dx,dy为0/1，选用横或竖算子，同时为1时计算横和竖边缘，但效果不如分别计算再相加好，ksize=a为卷积核边框大小

cv2.convertScaleAbs(image):将image取绝对值并转换为8位数

(cv2.addWeighted(sobelx1,1,sobely1,1,)

cv2.Scharr(image,ddepth,dx,dy,ksize):用Scharr算子计算，会有负数

cv2.Laplacian(image,ddepth):用Laplacian算子计算，会有负数

cv2.pyrDown(image):高斯下采样。图像变为1/4(与高斯核卷积后去除偶数行和列)

cv2.pyrUp(image):高斯上采样，图像扩大4倍(先将图像每个像素点的右、右下、下三个方向填充0，然后使用高斯核卷积获得近似值)

image-cv2.pyrUp(cv2.pyrDown(image)):利用高斯下、上采样进行边缘提取。下采样后再上采样与原图相比略缩小

cv2.Canny(image,minVal,maxVal):Canny边缘提取。minVal和maxVal(0-255)越小保留的细节越多

**模板匹配：**

list,array=cv2.findContours(image,mode,method):轮廓提取。list为所有轮廓组成的列表，包括很多没用的轮廓和图像边界框；array为每条轮廓对应的属性，image应先用处理为二值图。mode：cv2.RETR\_EXTERNAL只检测外轮廓，cv2.RETR\_TREE检索所有轮廓并重构嵌套轮廓的层次；method：cv2.CHAIN\_APPROX\_SIMPLE对于直线轮廓只保留轮廓终点

cv2.drawContours(image,contours,q,(a,b,c),w):将轮廓绘制到image中，会改变image。列表contours为轮廓信息；q为-1时显示所有轮廓；(a,b,c)为BGR通道表示绘制轮廓的颜色；w表示绘制线条的粗细

cv2.contourArea(list[a]):计算轮廓list[a]的面积

cv2.arcLength(list[a],True/False):计算轮廓list[a]的周长，True表示曲线是封闭的

x,y,w,h=cv2.boundingRect(list[a]):用矩形边框包含第a个轮廓

arrary1=cv2.matchTemplate(image,template1,method):模板匹配，返回各板块数组。method：cv2.TM\_SQDIFF平方差匹配法，值越接近0越好；cv2.TM\_CCORR相关匹配法，数值越大越好；cv2.TM\_CCOEFF 相关系数匹配法，1最好，-1最差；cv.TM\_SQDIFF\_NORMED归一化平方差匹配法；cv2.TM\_CCORR\_NORMED归一化相关匹配法；cv2.TM\_CCOEFF\_NORMED归一化相关系数匹配法

cv2.TM\_SQDIFF\_NORMED归一化平方差匹配，cv2.TM\_CCORR\_NORMED归一化相关匹配，cv2.TM\_CCOEFF\_NORMED归一化相关系数匹配

min\_val,max\_val,min\_loc,max\_loc=cv2.minMaxLoc(arrary1):得到arrary1中最小值、最大值和其位置，min/max\_loc中存放模板左上角的(列，行)

**获取关键点：**

sift=cv2.xfeatures2d.SIFT\_create()：调用sift特征提取

keypoints=sift.detect(image)：获取图像的关键点(建议用灰度图)

img2=cv2.drawKeypoints(image,keypoints,img2)：用彩色小圆圈绘制关键点，image为原图，img2为绘制关键点的图

**坐标变换：**

M=cv2.getAffineTransfrom(points1,points2)：得到Affine类型变换矩阵。Points1/2要至少3组对应的坐标

cv2.warpAffine(image,M)：将image进行仿射变换

M=cv2.getAffineTransfrom(points1,points2)：得到仿射变换矩阵。Points1/2要至少3组对应的坐标

cv2.warpAffine(image,M)：将image进行M仿射变换。可实现旋转，平移，缩放；变换后的平行线依旧平行

M=cv2.getPerspectiveTransform(points1,points2)：得到透视变换矩阵。Points1/2要至少4组对应的坐标

cv2.warpPerspective(image,M)：将image进行M透视变换。可保持直线不变形，但是平行线可能不再平行

**import sklearn(机器学习库)**

(pip install scikit-learn)

sklearn.metrics.explained\_variance\_score(A,B)：期望方差

sklearn.model\_selection.train\_test\_split()：划分数据集

**model=sklearn.neighbors.KNeighborsClassifier(n\_neighbors,weights,algorithm,leaf\_size,metric,p)：初始化K近邻模型。**n\_neighbors是KNN的k值默认为5：k值较小时训练误差小但泛化误差大，容易受噪声影响，k值较大时反之，可以采用交叉验证法选取k值；weights是权重：uniform均等权重，distance距离近的点权重影响较；algorithm：auto自动选择，brute是线性扫描,训练集大时很耗时，kd\_tree构造kd数存储数据，维数小于20时效率高，ball\_tree克服了kd树高纬失效；leaf\_size默认30：构造kd树和ball树的大小，这个值会影响树构建的速度和存储大小；P默认为2：p=2是欧氏距离，P=1是曼哈顿距离，曼哈顿距离一般会比欧氏距离长,但计算速度快,不会有误差

model.fit(train\_x,train\_y):训练模型

model.predict(test\_x):用模型预测

model.score(test\_x,test\_y):求准确率

**model=sklearn.cluster.KMeans(n\_clusters=8,max\_iter=300,n\_init=10)：初始化K均值模型。**n\_clusters是生成聚类数；max\_iter是执行一次算法最大迭代数；n\_innit是用不同聚类中心运行的次数，最终用inertia指标选出最优结果

model.fit(train\_x)：训练模型

model.labels\_：显示聚类结果

**model=sklearn.decomposition.PCA(n\_components,copy)：PCA主成分分析法。**基于方差最大化。n\_components是要变成的维度,用‘mle’表示采用mle算法自动决定变成的维度；copy默认为True,表示训练数据时使用备份,若为False会改变原始数据

model.fit(train\_x)：训练模型

model.explained\_variance\_：降维后各主成分方差,越大证明越重要

model.explained\_variance\_ratio\_：降维后各主成分方差占比

model.transform(train\_x)：降维后的数组

**model=sklearn.svm.SVC(C=1.0,kernel=’rbf’,cache\_size=200,max\_iter=-1,)：支持向量机分类。**C为惩罚参数：C越大准确度越高但泛化能力越弱；kernel为核函数：‘linear’线性核函数，’poly’多项式核函数，’rbf’高斯核函数；’sigmoid’sigmoid核函数；cache\_size为指定训练需要的内存；max\_iter为最大迭代次数：-1表示无限制。

model.fit(train\_x)：训练模型

model.predict(test\_x):用模型预测

**python中自带库**

**import os:**

os.listdir(文件夹路径)：读取文件夹中所有文件的相对路径（带格式）。在windows下读取的文件路径是自动排序好的，但linux中顺序是乱的。windows中读取路径可以用r’\’，linux不能识别

os.getcwd()：当前文件的相对路径

os.makedirs(path)：创建目录

os.chdir(path)：改变当前目录到path

os.remove(path)：删除位置为path的文件

os.rename(path,更改后名称)：更改文件名

os.path.abspath(‘’)：当前文件所在目录。’../’上级目录

os.path.join(A,B,C)：将A,B,C路径拼接起来，会自动加/

os.path.exists(path)：判断路径是否存在返回布尔值

os.path.isdir(path)：判断路径是否为目录

os.path.isfile(path)：判断路径是否为文件

os.environ：获取系统当前的环境变量，返回一个字典，键为环境变量名

**import math**

math.cos(a)：cos函数，a=0-1\*math.pi。math.pi为

**import pathlib**

pathlib.Path类似os.path的路径操作，但操作类型是WindowsPath

pathlib.Path(str1)：将路径转换为WindowsPath类型

pathlib.Path.cwd()：当前文件所在目录(需要用str()转换为字符串类型)

pathlib.Path.home()：用户的home根目录

pathlib.Path.exists(WindowsPath1)：判断路径是否存在返回布尔值

WindowsPath1 / WindowsPath2：相当于str1+’/’+str2

WindowsPath1.mkdir(parents=False)：创建路径文件。parents=True时会依次创建路径中间缺少的文件夹

文件名.rename(新文件名可带路径)：如果不带路径则直接更改文件名，如果带路径则是移动到目标路径下再更改文件名

**import sys(运行管理、与外部交互)**

sys.path:存放路径的列表，系统导入库函数时会在路径列表中的路径中找

for lines in sys.stdin：可以不断读取输入(输入为一行或多行，多行时处理较为复杂)

line=sys.stdin.readline().strip()：输入为多行时，读取第一行

line=list(map(int,sys.stdin.readline().strip().split()))：完整处理

**import base64：二进制加密传输格式**

base64.b64encode(字节类型数据)：将字节类型转为base64编码

base64.b64decode(base64编码)：将base64解码转为字节类型数据

**常用情况：**

1，with open(图片位置,’rb’,encoding='utf-8') as f: //a=f.read：读取为字节类型

//a=base64.b64decode(传输信息)：base64编码转为字节类型

2，b=np.frombuffer(a,dtype=np.uint8)：字节类型转为一行np数组

3，c=cv2.imdecode(b,cv2.IMREAD\_COLOR)：一行np数组转为BGR图片数组

//c=cv2.imread(图片位置)：读取图片为BGR数组

4，d=cv2.imencode(‘.jpg/png’,c)[1]：BGR图片数组转为一行np数组

5，e=d.tostring()：一行np数组转为字节类型

6，base64.b64encode(字节类型数据)：字节类型转为base64编码

**import collections**

Collections.Counter(array)：统计array中各元素个数

dict=Collections.OrderedDict()：建立有序字典

**import copy**

copy.deepcopy(list):完全复制一份list

**import argparse(python自带的命令行参数解析包)**

parser=argparse.ArgumentParser(description='提示')：创建参数解析器

parser.add\_argument(‘--参数名’,default,type,choices=[],action,required,help)：添加参数。default为不传入参数时的默认值；type是命令行参数会被转成的类型如type=str；有choices时固定选用的参数；action=‘store\_true’时如果命令行传入参数如--A，则A的值为True，否则默认为False，‘store\_false’与之相反；required=True时必须从命令行传入参数；help为提示信息

args=parser.parse\_args()：解析参数。args中包含所有定义的参数，args的类型为namespace

在终端使用’cd’命令进入文件所在目录，输入‘python 文件名.py --参数名 值 --参数名 值...’传入参数并运行程序

python -m torch.distributed.launch --master\_port 9999 --nproc\_per\_node 2 文件名.py --device 0,1 --参数名 值...：启用pytorch的分布式训练。master\_port为随意一个未被占用的端口即可，nproc\_per\_node为使用GPU数，device为使用的GPU编号

**import xml.etree.ElementTree as ET**

root=ET.parse(文件路径.xml).getroot()：定义文件对应的root工具

root.find/findall(‘标签’)：读取xml文件中的/所有的’标签’

root.find(‘标签’).text：将读取的标签以正常形式显示

for object in root.findall('object')：遍历xml文件中所有的'object'标签，并定义object工具(同root)，可以继续使用object.find(‘标签’)

object[i][j].text：读取标签object中第i个子标签的第j个内容

**import time:**

time.time()：当前系统时间

**import warnings**

warnings.filterwarnings('ignore')：忽略此条语句之后的报错

**import json**

with open(path\_label+'/'+dir\_json[i],'r',encoding='utf8') as f: //load=json.load(f)：解析json文件

with open('保存的文件名',‘w’,encoding=’utf-8’) as f: /json.dump(‘待转换的字典’, f, ensure\_ascii=True)：转字典为json文件并保存。ensure\_ascii=True时会将中文转换为ascll码便于直接阅读(占用更少的空间)

**标准COCO数据集格式**为json，在数据转换为json格式前也需要变成对应的标准字典格式，COCO中annotations保存的坐标为xmin,ymin,w,h：

{‘info’,’categories’,’images’,’annotations’}

info{‘year’,’version’,’description’,’contributor’,’url’,’date\_created’}



categories[{‘id’,’name’}]

images[{‘id’,’file\_name’,’width’,’height’}]

annotations[{‘id’,’category\_id’,’image\_id’,’box’:[xmin,ymin,w,h],’area’}]

**import jsonlines：jsonl是json的扩展**

with open(path\_label+'/'+dir\_json[i],'r',encoding='utf8') as f: //load=[\_ for \_ in jsonlines.Reader(f)]：解析jsonl文件

**import random**

random.randint(a,b)：生产a到b区间的随机整数

random.choice([...])：随机选取列表中一个数

**import concurrent**

**import datetime**

datetime.datetime.now()：获取当前时间(年月日时分秒)，str()转字符串

datetime.datetime.now().date()：获取当前时间(年月日)

**import threading**

**import thop：计算模型FPS**

in\_ = torch.rand(1,3,640,640,dtype) ：创建输入张量

flops, params = profile(model, inputs=(in\_, ))：计算FPS和参数量

**from concurrent.futures import ThreadPoolExecutor：多线程**

**import functools：固定常量**

def A(x,y):...：创建函数

A\_partial=functools.partial(A,y=1)：固定A函数的y变量为1。使用A\_partial函数时就不能传入y参数了。使用多线程时对于固定的参数可以使用partial固定，但要注意固定的参数在函数中要在待传入参数的后面

with ThreadPoolExecutor(a) as executer:：创建多线程。a为线程个数，不指定a时会自动采用最大值

//executer.map(A,[x1,x2...])：与map函数类似，先将变量列表分配到多个线程中，在各线程中依次将变量传入A函数并返回map对象(可再加list变成列表)

**import re**

正则表达式测试网站：<https://c.runoob.com/front-end/854/>

(^| )(13[0-9]|15[0-9]|18[0-9])\d{8}($| )：电话号码正则表达式。优先级()>其他>|；|表示或；^表示字符前面不能有其他字符；$表示字符后面不能有其他字符；(^| )表示除了空格不能有其他字符；数字n表示该位置要为数字n(也可以为汉字)；[0-9]表示该位置要为0-9中一个；\d{m-n}表示后面要有任意n-m个数字

regex=re.compile(正则表达式)：定义正则匹配函数

result=regex.findall(字符串内容)：找到所有匹配的值返回到列表中

result=regex.match(字符串内容)：匹配到任意内容就返回。没匹配到返回None，匹配到使用result.group()得到匹配到的内容，result.span()得到元组(起始字符下标，结束字符下标)

**其他库**

**from PIL import Image：**

Image.open(路径):读取图片；image.save(路径):保存图片

image.format:看格式；image.size:看尺寸；image.mode:看色彩模式

plt.imshow(image对象/numpy数组):显示图片

image.convert(色彩模式):1是二值，L是灰度，P是8位彩色，RGB是24位彩色，RGBA是32位彩色，CMYK是CMYK彩色，YCbCr是YCbCr彩色，I是32位整形灰度，F是32位浮点灰度

np.array(pic1):将图片转化为数组

image.resize((x,y)):缩放图像

image.transpose(旋转方式):旋转图像

image.crop((x1,y1,x2,y2)):裁剪图像

**import requests:**

code1=requests.get(‘网址’):获取网页代码;code1.text显示代码

**from mpl\_toolkits.mplot3d import Axes3D**

Axes3D.scatter(x,y,z):三维散点图

Axes3D.plot\_surface(x,y,z):三维表面图

Axes3D.plot\_wireframe(x,y,z):三维线框图

**import psycopg2**

建立和操作数据库PostgreSQL

(pip install psycopg2-binary安装)

connect=psycopg2.connect：建立一个数据库connect

cursor=connect.cursor()：建立一个光标执行数据库操作

**import obs：从网站上下载数据**

(pip install esdk-obs-python安装)

obs=obs.ObsClient(access\_key\_id,secret\_access\_key,server)：配置

obs.getObject(容器名称,图片服务器路径,保存位置,loadStreamInMemory=False)：从服务器下载图片

image=obs.getObject(容器名称,图片服务器路径,保存位置,loadStreamInMemory=True)：从服务器下载图片到本地缓存而不直接保存，此时会自动忽略保存位置

image=image.body.buffer：得到图片的字节类型数据

image=np.frombuffer(image,dtype=np.uint8)：将字节类型一行数组

image=cv2.imdecode(image,cv2.IMREAD\_COLOR)：将一行数组转为BGR图片

**import pymongo：一种数据管理库**

**import tabulate：绘制表格**

tabulate.tabulate(...)：绘制表格

**tqdm：显示进度条**

for i in tqdm.tqdm(range(n))：显示循环的进度条

tqdm\_show=tqdm.tqdm(iterable,total,postfix=dict,mininterval=0.1)：定义显示方法。iterable为可迭代对象，或者使用total指定长度；postfix为传入的说明(dict格式)；mininterval为最小显示间隔

//tqdm\_show.set\_postfix({'当前loss':loss\_batch.item()})：实时添加显示信息

//tqdm\_show.update(1)：更新进度条，进度条总长度为设定的total，1为更新的进度

list(tqdm.tqdm(多进程,total=len(长度)))：对于多进程等情况如果不显示进度条要加total参数，加list是让进度条会显示加载进度

**import albumentations：图片处理与增强，需要opencv库**

<https://albumentations.ai/docs/getting_started/image_augmentation>

transform=albumentations.Compose([...])：定义转换的框架。将其他执行的函数依次放入框架中

//image=transform(image=image)[‘image’]：执行框架

albumentations.LongestMaxSize(max\_size)：等比缩放，指定最大边长

albumentations.SmallestMaxSize(min\_size)：等比缩放，指定最小边长

albumentations.Normalize(max\_pixel\_value=255.0,mean=(0.485,0.456,0.406),std=(0.229,0.224,0.225))：归一化、减均值、除以方差。默认为BGR

albumentations.PadIfNeeded(min\_height=320,min\_width=320,border\_mode=cv2.BORDER\_CONSTANT,value=(126,126,126))：向图片四周填充value

albumentations.GaussianBlur(blur\_limit=(3,3),sigma\_limit=0,p=0.5)：高斯模糊。blur\_limit为卷积核大小；sigma\_limit为标准差；p为使用的概率

albumentations.GaussNoise(var\_limit=(10,50),p=0.5)：高斯噪声。var\_limit为噪声方差范围；p为使用概率

**import yaml**

(pip install pyyaml安装)

**import wandb：训练可视化**

Weights & Biases是一个类似于tensorboard的在线模型训练可视化网站，wandb是Weights & Biases的辅助库，放到代码中可实时上传到网站观看。除此之外还会保存日志信息到当前目录wandb下。使用时wandb库时用VPN可能会影响

使用教程：<https://docs.wandb.ai/guides>

先注册Weights & Biases账号：<https://wandb.ai>

安装wandb库后在命令终端使用wandb login，复制Weights & Biases网站个人账户中的API key粘贴到命令终端指定位置就可以完成登录。Project Defaults中可以设置保存到哪个项目中(个人/小组的)

wandb\_run=wandb.init(project,name,config)：初始化，在网页中可看到。project为网站中项目的名称；name是单次训练的名称，一个项目中的所有单次训练可以合在一起观察；config是记录一些配置信息，定义了argparse后可以直接config=args

wandb\_run.log({‘metric/val\_loss’:val\_loss},...)：添加数据，’/’前面为网页中的集合名，后面为集合中的各图表名。一般放在每轮循环中，在wandb网址上会自动生成图像。CPU和GPU的使用信息会自动获取。每一次使用log时所有图x的都会+1，因此一轮中只能有一个log

wandb\_image=wandb.Image(image,boxes=None)：将图片变为可添加的对象。image为rgb通道的numpy数据

//boxes={"predictions":{"box\_data":[box1,box2,...],"class\_labels":wandb\_class\_name}}：给图片添加边框

//box1={"position":{"minX":0.1,"minY":0.2,"maxX":0.3,"maxY":0.4},"class\_id":0,"box\_caption":wandb\_class\_name[0],"score":{"score":b}}，position为相对坐标，class\_id和box\_caption用于给边框加标签，有score时可以根据数值筛选框

//wandb\_class\_name={ 0:'name0',1:'name1',... }

//使用wandb\_run.log可添加wandb\_image或[wandb\_image1,wandb\_image2,...]，添加多张图片会加上筛选功能和index筛选项，每一次wandb\_run.log上传都为一个step

**import fiftyone：目标检测数据集可视化工具**

官方教程：https://docs.voxel51.com/index.html

可以选择本地数据(图片分类、目标检测)并后以浏览器的方式查看，会自动去除空标注的图片和找不到图片的标注，数据缓存为文件.fiftyone。需要pycocotools库

(pip install fiftyone)

(pip uninstall fiftyone fiftyone-brain fiftyone-db)

dataset=fiftyone.Dataset.from\_dir(name,labels\_path,data\_path,dataset\_type=fo.types.COCODetectionDataset)：查看数据。name为自定义名称，默认为当前的时间日期；labels\_path为coco标签位置(为空时也可正常运行)，如果设定了data\_path，coco标签可以为相对路径否则要为绝对路径

dataset.persistent=True：保存缓存，方便之后查看(可以设置为False并直接加载空标签，以快速启动fiftyone)

/ session=fiftyone.launch\_app(dataset)：启动fiftyone查看dataset

/ session.wait()：让网页等待，才能看到。如果网页关了要重新启动

如果在A服务器上启动程序，则网址为”A服务器公网地址:端口号”，端口号会在运行程序后给出，一般为5151

**import doccano：文本分类标注工具**

官方项目：<https://github.com/doccano/doccano>

**import Cython**

Cython可以编译python语言，提高运行速度

**import pycocotools**

windows安装较复杂，linux可直接安装

**import onnx,onnxsim：压缩简化onnx模型**

(pip install onnx-simplifier(同时会安装onnx))

model=onnx.load(onnx位置)：加载onnx模型

onnx.checker.check\_model(model)：检测onnx模型，不报错就是没问题

model\_simplify,check = onnxsim.simplify(model)：压缩简化onnx模型

onnx.save(model\_simplify,保存位置)：保存模型

**import onnxruntime：onnx模型转化和推理**

官方项目：https://github.com/onnx/onnx

(pip install onnxruntime：安装cpu版本)

(pip install onnxruntime-gpu：安装gpu版本，cpu和gpu都可以用)

onnx是机器学习模型加速库，一般将tensorflow/pytorch模型转换为onnx，然后从onnx转换为tensorrt。onnx模型可以用netron软件查看。onnx模型转出后输入的形状就确定了，并且不再依赖模型代码可直接加载运行(pt模型加载需要模型代码)

**推理：**

在电脑支持的情况下，CPU/GPU上都可以使用float32/float16推理。输入数据要为np.array

session = onnxruntime.InferenceSession(onnx位置,providers)：加载模型和框架。providers=['CPUExecutionProvider']/['CUDAExecutionProvider']使用CPU/GPU推理

input\_name = session.get\_inputs()[0].name：获取网络输入名称。[0].name为第1个输入的名称

output\_name = session.get\_outputs()[0].name：获取网络输出名称

pred = session.run([output\_name], {input\_name: np数据})：推理

**import netron：查看onnx模型结构**

netron有还有专门的软件，可以直接打开.onnx文件查看网络结构

netron.start(onnx)：在跳出的网页中查看onnx模型的网络结构

**import tensorrt：针对英伟达GPU推理**

下载：https://developer.nvidia.com/nvidia-tensorrt-8x-download

官方项目：<https://github.com/NVIDIA/TensorRT>

使用函数：<https://docs.nvidia.com/deeplearning/tensorrt/archives/tensorrt-840-ea/api/c_api/index.html>

(安装要从NVIDIA官方下载对应cuda、cudnn版本的压缩包)

需要先安装cuda和cudnn，下载tensorrt版本要对应，只需要压缩包中的bin、include、lib文件，然后添加lib文件路径到系统路径中(windows为：系统->高级系统设置->环境变量->系统变量->Path中加入。linux为：sudo ldconfig lib位置)，然后找到对应版本的whl文件使用pip install .whl。bin中是官方提供的onnx转trt程序

tensorrt是英伟达官方针对自己的硬件设备面向AI工作者推出的一种部署方案，可以改善网络的延迟、吞吐量以及效率。对于trt模型，批量预测相对单张预测只有在小模型上有一点优势，大模型上差不多，甚至单张预测更快，反而单张预测占显存更少

**onnx转trt：使用bin中的trtexec**

linux需要添加环境变量export LD\_LIBRARY\_PATH=$LD\_LIBRARY\_PATH:lib位置。命令终端输入trtexec：会看到使用的提示信息

trtexec --onnx=best.onnx --saveEngine=best --useCudaGraph ...

转换过程中有很多提示信息，可以解决大多数问题。转换后会进行速度测试。不指定输入形状时默认为单批量预测(推荐)

# --onnx=onnx模型位置

# --saveEngine=trt模型保存位置

# --noTF32：禁用float32精度

# --fp16：启用float16精度(通常使用fp16)

# --int8：启用int8精度(精度下降较为严重)

# --best：开启所有精度(有的模型是混合精度的)

# --device=0：使用的GPU号码，默认为0

# --useCudaGraph：尝试使用cuda图

**使用tensorrt推理：**



input为输入、h\_input为锁存、d\_input为显存；output同理

使用cuda延迟加载能够提升性能，但需要较高的cuda版本

**import pycuda.driver as cuda：**python的cuda接口，cuda由c和少量c++组成，python要通过pycuda接口来使用cuda计算

**import pycuda.autoinit**：导入后会自动进行一些数据初始化和内存管理

logger=tensorrt.Logger(tensorrt.Logger.WARNING)：创建日志记录器

with open(model\_path, "rb") as f,tensorrt.Runtime(logger) as runtime: //model=runtime.deserialize\_cuda\_engine(f.read())：读取模型

model.num\_bindings：模型的接口总数(输入口和输出口)

for binding in model:：binding为模型接口的字符串名称

//model.get\_tensor\_mode(binding)：判断接口是否为输出返回bool值

//tuple(model.get\_tensor\_shape(binding))：获取接口的形状

//tensorrt.nptype(model.get\_tensor\_dtype(binding))：获取接口的数据类型并转为np的字符串格式

tensorrt.volume(形状:model.get\_tensor\_shape(binding))：计算该形状的一维长度。如输入(1,3,640,640)得到

h\_input=np.zeros(长度:(tensorrt.volume(model.get\_tensor\_shape(binding))),dtype=np.float32)：获取输入的形状(一维)。h\_output同理

h\_input.nbytes：获取h\_input的字节数。float32为，16为2

d\_input=cuda.mem\_alloc(h\_input.nbytes)：分配显存空间。d\_output同

stream = cuda.Stream()：创建cuda流

bindings=[int(d\_input), int(d\_output)]：绑定显存输入输出

context = model.create\_execution\_context()：创建模型推理器

cuda.memcpy\_htod\_async(d\_input, h\_input, stream)：将输入数据从CPU锁存复制到GPU显存。h\_input为输入的数据(一维)

context.execute\_async\_v2(bindings=bindings,stream\_handle=stream.handle)：执行推理

cuda.memcpy\_dtoh\_async(h\_output, d\_output, stream)：将输出数据从GPU显存复制到CPU锁存。h\_output为推理输出的结果(一维)

stream.synchronize()：同步线程

**import insightface**

官方项目：<https://github.com/deepinsight/insightface>

(安装insightface时会自动安装很多依赖库)

运行后下载的模型文件位置在本地用户中

人脸识别库，需要onnxruntime库。insightface库中不包括一些模型文件，因此运行一些函数时会下载模型文件到用户下的.insightface文件中

model=insightface.app.FaceAnalysis()：导入模型。未指定会自动下载默认的官方给定模型到本地用户下的.insightface，之后可直接调用

/ model.prepare(ctx\_id, det\_size=(a,a))：模型设置。ctx\_id=0为使用cpu推理；det\_size为推理时输入的图片尺寸，一般为(640,640)(实际输入图片大小可以不为640，会自动调整)

faces=model.get(image)：获取图片中所有人脸框的信息。faces为列表，列表每个元素中包含多种信息，如人脸特征.normed\_embedding(512条)人脸位置.bbox(xmin,yxmin,xmax,ymax)

image\_draw=model.draw\_on(image, faces)：画出人脸框