1. 在添加系统环境变量时，是添加文件所在的目录，而不是具体的某个文件或者应用程序

PS的快捷键

Shift+滚轮

Ctrl+滚轮

Alt+滚轮

2021/1/19

Tensorflo为以后端的Keras框架 2DU-Net网络模型（端到端的模型）

步骤：10测试集 20验证集 70训练集

Cuda 与 cudnn安装成功

在win的命令窗口中，原生python与anaconda中的python都设置了环境变量，在调用时环境变量的优先级不同。

1.31

PET：正电子发射计算机断层显像

人工智能在核医学上的应用：

1. 在探测器水平上使用人工智能对图像重建的数据处理，包括与检测过程相关的不同物理过程(如衰减、散射)的校正。除了图像重建步骤之外，人工智能还可以用于不同图像的处理过程，包括去噪、分割和融合。最后，人工智能可以用于基于图像信息的建模，这将有助于实现基于图像的个性化医疗的决策。
2. 在检测方面，最近的工作包括通过使用CNN提高PET图像分辨率，改善具有大像素化晶体 [42] 的PET扫描仪的噪声特性，以及直接从配对符合数字化探测器波形中估计飞行时间 [43] 。将深度神经网络集成到迭代图像重建过程中，可以提高最终的图像质量 [44,45] 。深度学习方法已经被研究人员提议用于PET/CT和PET/MR的衰减校正和配准，并已被证明能够生成高精度的衰减图 [46-50] 。在同样的背景下，深度学习已经可用于改善具有飞行时间PET数据的衰减校正和放射性活度的最大似然重建（MLAA） [51] 。去噪处理就是成功使用深度学习技术中一个最受欢迎的图像处理应用之一，例如从低剂量图像生成足够剂量PET图像 [52] 或直接对重建的PET图像进行滤波 [29] 。（**宋少莉**）

2.4

**1.线性函数与非线性函数**

线性函数：指那些线性的函数，线性函数是指在坐标轴中，过原点的直线的函数

非线性函数：即指函数图像不是一条直线的函数

1. **神经网络中的激活函数：**

（激活函数必须使用非线性的函数，使用线性函数无法发挥多层网络带来的优势）

* 1. Sigmoid函数：1/1+exp(-x)
  2. ReLU函数：线性整流函数

h(x)={x(x>0);0(x<=0)}

代码：

Def relu(x):

Return np.maxium(0, x)

* 1. array 函数的使用 多维数组的生成

2.6

Pycharm的使用：  
在工程下新建文件：ctrl+insert

运行文件：shift+F10

调用表情包：ctrl+shift+B

设置断点（让程序在任意位置停下）：设置完断点后，先调试程序，程序会在第一个断点处停留，在Console窗口中会显示出程序运行到停留断点处变量的值，点击按钮进入下一个断点或用快捷键（ctrl+F9）

2.19

数组上的维度与矩阵的行数不一样

3.2

Jupyter notebook学习

总结：

打印格有数字才代表执行了代码，\*表示代码正在执行

自动补全的操作

查看方法的使用

数组的使用

总结：

1. 引入numpy，ndarray多维数组对象
2. 多维数组的生成，a = numpy.array([value])
3. 多维数组的切片，利用索引号，逐层的提取数据，由多维到一维，由一维到值：arr[2][1] 第二行第1列的数；多为数组的切片arr[: 2, 3 : 4] 先行后列，无索引好则表示前几行（如前2行）

3.3

Github的学习

1.基础的概念：

1.1目的：借助Github托管项目代码

1.2仓库（repository）：存放项目代码，每个项目对应一个仓库

1.3收藏（star）：收藏项目方便下次查看

1.4 Fork：复制克隆项目，复制后独立存在。在自己的仓库中直接生成了别人的项目，能在此项目中自己做更改，向之前的仓库创建者发出请求（pull request），别人同意后可选择是否合并更改。

1.5 Pull Request：发起请求，基于Fork的。别人向原项目的创建者发表请求，原创者可以查看请求的代码，并且测试觉得ok，就会接受请求。

1.6关注（watch）：如果你关注了某个项目，当项目更新则会接收到通知

2.Github界面

2.1主页

2.2 仓库主页

2.3 个人主页

3.建立仓库

1.仓库名一般为项目名

2.在仓库下多一个readme的说明文件来详细描述项目

1. 仓库的管理
   1. 新建仓库
   2. 在仓库中提交文件，修改文件
   3. 在仓库中搜索文件 快捷键T
2. Git-hub Issue
   1. 在Git-hub主页中显示出issue，操作解决issue

Fork到仓库的文件，进行更改后，原仓库的文件不会同步。

如何使原仓库同步：

Fork仓库发出请求，原仓库就能收到请求，并查看测试后考虑是否更改。

1. 开源项目贡献流程
   1. 通过新建Issue，提交使用问题
   2. Pull request

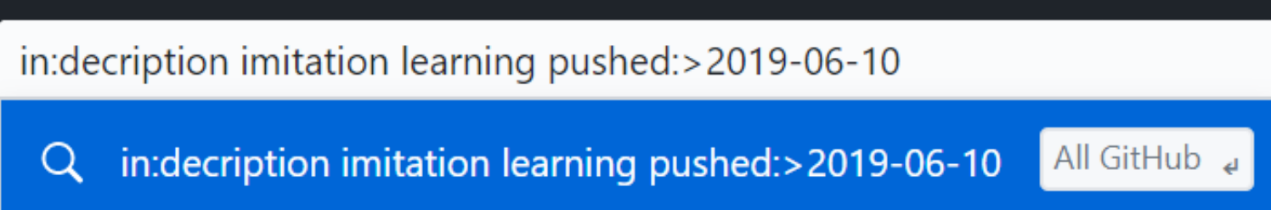
Fork项目

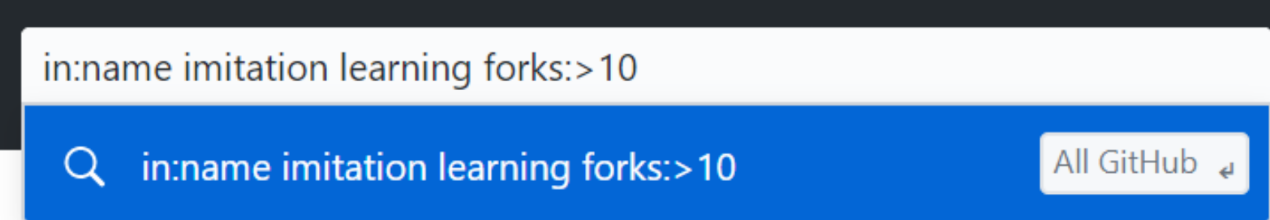
。。。。

1. 在Git-hub上找源码

通过in 限制符







3.4

待解决的问题

编码？

Numpy.Sqrt（）函数：返回数的平方根 square：平方米 平方

举国维艰之时责任更为重要

Numpy中的random模块的详解：

Rand（数组的类型（如：（3,2）））：随机生成三行两列的随机数组

Randn（数组的类型）：具有正态分布的随机数

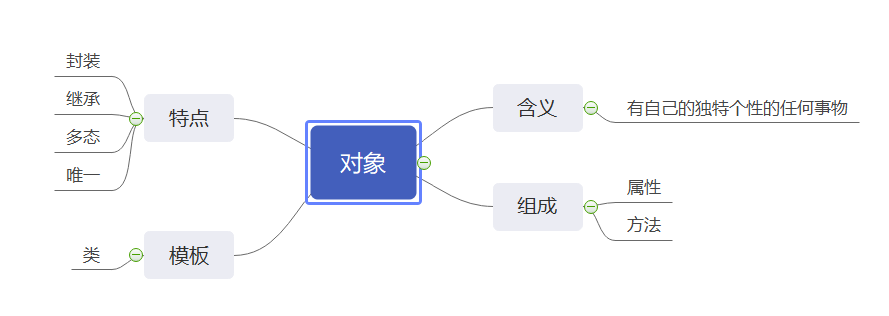
numpy.random.randint(low, high=None, size=None, dtype='l')：生成大于等于low，小于high的整数数组

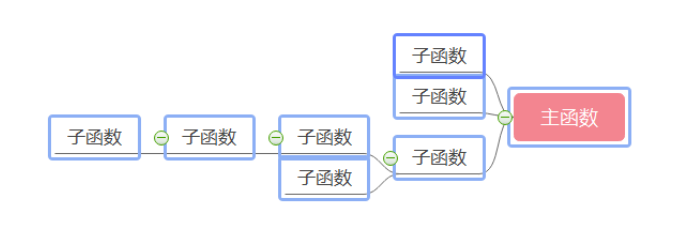
（如：np.random.randint(2,high=10,size=(3,2)）

np.random.[random\_integers](http://docs.scipy.org/doc/numpy/reference/generated/numpy.random.random_integers.html#numpy.random.random_integers)() low与 high的闭区间中的整数数组

**拓展**

面对对象：

　面向对象就是构成问题事务分解成各个对象，建立对象的目的不是为了完成一个步骤，而是为了描叙某个事物在整个解决问题的步骤中的行为。

面向过程：

面向过程不同于面向对象，面向过程分析出解决问题所需要的步骤，然后用函数把这些步骤一步一步实现，使用的时候一个一个依次调用就可以了。

**四、总结**

**面向过程：**

**优点：性能比面向对象高，因为类调用时需要实例化，开销比较大，比较消耗资源;比如单片机、嵌入式开发、 Linux/Unix等一般采用面向过程开发，性能是最重要的因素。**  
**缺点：没有面向对象易维护、易复用、易扩展**

**面向对象：**

**优点：易维护、易复用、易扩展，由于面向对象有封装、继承、多态性的特性，可以设计出低耦合的系统，使系统 更加灵活、更加易于维护   
　　　　缺点：性能比面向过程低**

**使用数组进行面向数组的编程**

**在jupyter notebook中使用数组的可视化matplotlib模块时，要先使用魔法函数将ipython与matplotlib连接，才能显示出图像。**

**数组的逻辑操作符：**

**Np.Where（条件，true时返回，false返回）**

**数组是在Numpy下的一种实例化，也继承了numpy的方法：**

**在调用数学函数时，可以直接调用实例化的方法如：arr.mean()**

**或者使用顶层的Numpy函数 如：np.mean(arr)**

**多维数组中轴的理解：**

**二维数组：沿行向下 axis =0 为第一轴 沿列横穿为第二轴 axis=1  
sort mean sum row column**

**Unique：返回数组中的唯一值并排序**

**Any all 验证数组中的布尔值**

**Intersection：intersect1d（x,y）的交集并排序**

**Union1d（x,y）的并集，并排序**

**In1d（x,y）：x是否在y中，返回布尔值**

**Setdiff1d(x,y):在x中但不在y中的x的元素**

**Setxor1d（x,y） 在x或y中，但不同属于x,y**

**是对数组中的所有元素，返回一个一维的数组。**

**使用数组进行文件输入与输出：**

**Np.save np.load 数据文件格式为.npy**

**Pandas入门：**

1. **Series**

**Series数据类型：与array不同是有索引，且是一维的序列值，可以看成一个定长的有序字典，能适用于numpy的函数。**

**Series对象自身也有属性：实例化对象具有 name属性，用于设置列表的索引等。**

1. **DataFrame**

**DataFrame：矩阵的数据表，数据存储为一个二维块**

**在操作或检索列时（column），当dataframe对象中设置了column变量时可以直接用datafram.Column。而datafram【’column’】则均有效。**

**删除列 del**

**由列检索出的Serise，在操作时会映射到dataframe中，为了提取出列的数据，因用Series.copy（要copy的对象）**

**3.15 OS模块**

**OS模块的作用：**OS模块提供了多数操作系统的功能接口函数。当os模块被导入后，它会自适应于不同的操作系统平台，根据不同的平台进行相应的操作，在python编程时，经常和文件、目录打交道，这时就离不了os模块

**常用的OS模块命令**

1. **OS.getcwd（）（get current work directory缩写）：获取当前工作的目录**
2. **OS.listdir（path）（list directory缩写）——列出path目录下所有的文件和目录名（不包括子目录）。**

**注：只能向此函数中传入变量，不能传入值**

1. **OS.remove（path）——删除path指定的文件**
2. **Os.rmdir（path）——删除path指定的目录**
3. **Os.mkdir（path）——创建path指定目录 只能建立一层，要想递归建立用：os.mkdirs（）**
4. **Os.walk（path）——返回一个三元组（root，dirs，files），root-在正遍历的文件夹的本身的地址；dirs-该文件夹中所有的目录的名字；files-该文件夹中所有的文件**

**Tqdm模块：**是一个快速，可扩展的Python进度条，可以在 Python 长循环中添加一个进度提示信息，用户只需要封装任意的迭代器 tqdm(iterator)

**理解：给一个循环迭代器一个可视化的进度条**

**使用方法：1** from tqdm import tqdm

for i in tqdm(range(1000)):

*#do something*

Pass

Tqdm（list\数组）

**2.trange（一种简化的形式）**

**for i in trange（100）**

1. **在for循环外部初试化tqdm**

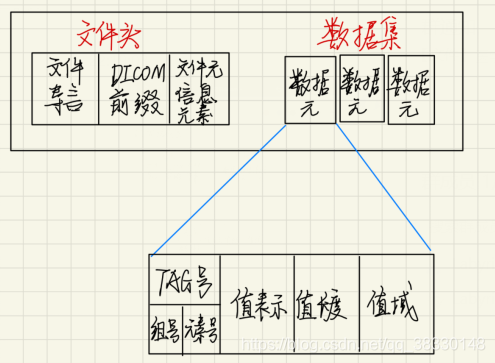
bar = tqdm(["a", "b", "c", "d"])

for char in bar:

bar.set\_description("Processing %s" % char)

endswith（“字符”）：用于判断字符串是否有“字符”后缀

**DICOM文件：**患者的医学图像以DICOM文件格式进行存储，其中包含了图像信息以及患者的**PHI**（protected health information，即姓名、性别、年龄等），以及产生图像的设备的相关信息。如下图所示，以dcm后缀结尾的文件即DICOM文件，其存储的信息为二进制格式。

**组成结构图：**

Matplotlib的使用手册：Matplotlib 是Python中类似 MATLAB 的绘图工具，熟悉 MATLAB 也可以很快的上手 Matplotlib

1.Plt.figure():生成一个figure图画对象，可以理解为一张画板

2.在拥有figure对象后，ax = fig.add\_subplot(参数) Eg:(111)

前两位表示几乘几的图纸，后面表示有几个图纸

1. 设置图纸的大小：ax.set(xlim=[],ylim=[],title=””,ylabel=””,xlabel=””)
2. 散点图：plt.scatter(x,y,color=””,marker(点的形状)=””)
3. np.random.seed(1)

x = np.arange(5)

y = np.random.randn(5)

fig, axes = plt.subplots(ncols=2, figsize=plt.figaspect(1./2))

vert\_bars = axes[0].bar(x, y, color='lightblue', align='center')

horiz\_bars = axes[1].barh(x, y, color='lightblue', align='center')

#在水平或者垂直方向上画线

axes[0].axhline(0, color='gray', linewidth=2)

axes[1].axvline(0, color='gray', linewidth=2)

plt.show()

1. 直方图：

np.random.seed(19680801)

n\_bins = 10

x = np.random.randn(1000, 3)

fig, axes = plt.subplots(nrows=2, ncols=2)

ax0, ax1, ax2, ax3 = axes.flatten()

colors = ['red', 'tan', 'lime']

ax0.hist(x, n\_bins, density=True, histtype='bar', color=colors, label=colors)

ax0.legend(prop={'size': 10})

ax0.set\_title('bars with legend')

ax1.hist(x, n\_bins, density=True, histtype='barstacked')

ax1.set\_title('stacked bar')

ax2.hist(x, histtype='barstacked', rwidth=0.9)

ax3.hist(x[:, 0], rwidth=0.9)

ax3.set\_title('different sample sizes')

fig.tight\_layout()

plt.show()

1. 饼图

labels = 'Frogs', 'Hogs', 'Dogs', 'Logs'

sizes = [15, 30, 45, 10]

explode = (0, 0.1, 0, 0) # only "explode" the 2nd slice (i.e. 'Hogs')

fig1, (ax1, ax2) = plt.subplots(2)

ax1.pie(sizes, labels=labels, autopct='%1.1f%%', shadow=True)

ax1.axis('equal')

ax2.pie(sizes, autopct='%1.2f%%', shadow=True, startangle=90, explode=explode,

pctdistance=1.12)

ax2.axis('equal')

ax2.legend(labels=labels, loc='upper right')

plt.show()

1. 等高线（轮廓图）

fig, (ax1, ax2) = plt.subplots(2)

x = np.arange(-5, 5, 0.1)

y = np.arange(-5, 5, 0.1)

xx, yy = np.meshgrid(x, y, sparse=True)

z = np.sin(xx\*\*2 + yy\*\*2) / (xx\*\*2 + yy\*\*2)

ax1.contourf(x, y, z)

ax2.contour(x, y, z)

**布局：**