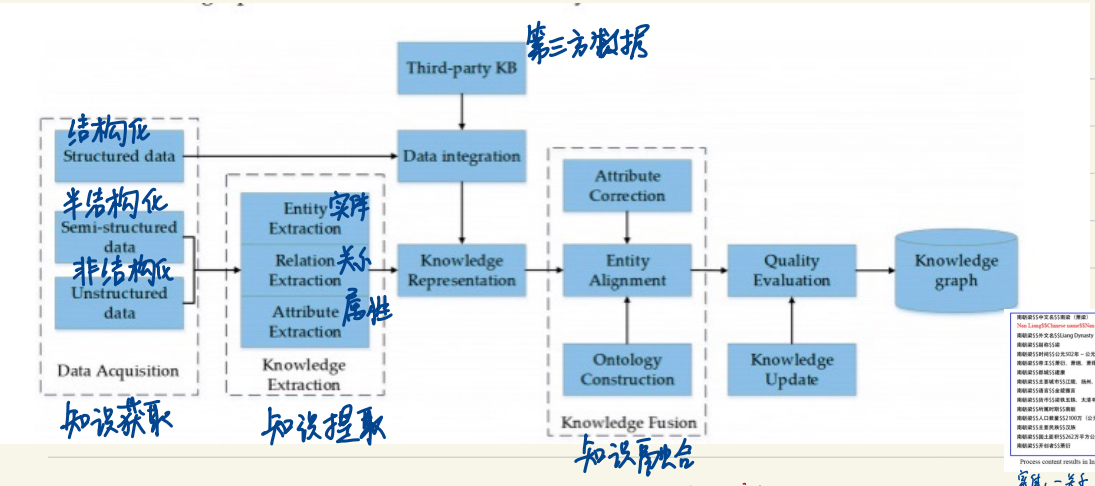
# NLP

# 基本用法速查

list → str : ''.join( )

分隔符 待转区域




案例1 - 关系 - 实体2

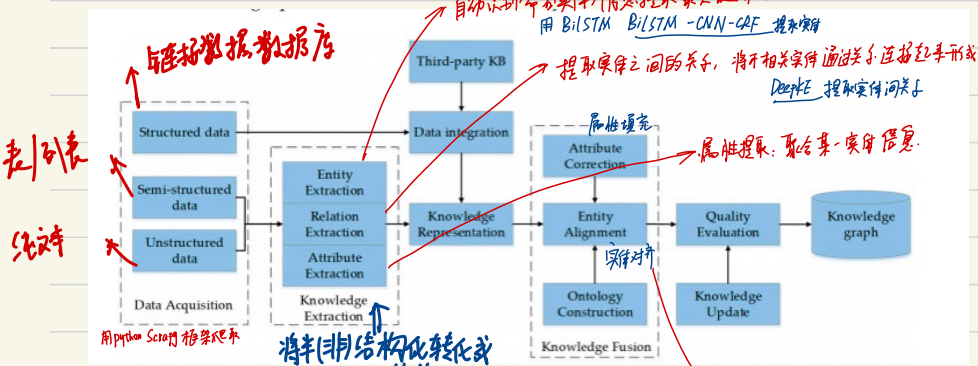


**Figure 2.** Baidu Encyclopedia data acquisition example (taking Nan Liang as an example), It is mainly divided into three parts. The first part is the Introduction part. This part is unstructured text data. After the extraction completed, the user named entity recognition and relationship extraction are performed. The second part is the Image part. This part mainly obtains the picture information. The obtained picture will be applied in the knowledge attribute query module of the knowledge graph system. The third part is the InfoBox part. This part is semi-structured data, which is mainly used to construct triples.

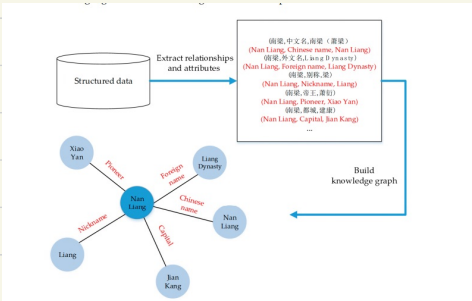



**Figure 4.** Structured data to graph database stored procedures.

Two principles in the construction of the knowledge graph:

每个节点由其存储编号表示

1. The type of each node is represented by its identifier. If the identifier is "dynasty", the type of node created is "dynasty".
2. In the entity-relation table, the storage format of the triples data is "Entity1-Relation-Entity2" or "Entity1-Attribute-Attribute Value".

三元组 { 实体1 - 关系 - 实体2
实体1 - 属性 - 属性值

BiLSTM - CNN - CRF

CBOW
词向量训练工具

上下文输入中预测
当前单词

1. ① 训练单词向量
   ② 提取句子字符向量
   ③ 得到每个单词字符级特征

# 对话系统

大的意图集合，比如音乐领域，视频领域

输出(领域, 意图, 槽位)
Domain Intent Slot

提取意图

```
Q = "订一张明天战狼的电影票"
intent = "订电影票"
slot = {
    "电影名": "战狼",
    "电影院名称": None,
    "时间": None,
    "数量": 1,
    "座位": None
}
A ="请问你想定哪个影院的呢？"
```

意图

槽位

槽位未满
追问

同构图：在图里面，节点的类型和边的类型只有一种的图，举个例子，像社交网络中只存在一种节点类型，用户节点和一种边的类型，用户-用户之间的连边。

异构图：在图里面，节点的类型+边的类型>2的一种图，举个例子，论文引用网络中，存在着作者节点和paper节点，边的关系有作者-作者之间的共同创作关系连边，作者-论文之间的从属关系，论文-论文之间的引用关系。

患者信息图 —判断→ 心脏病类别

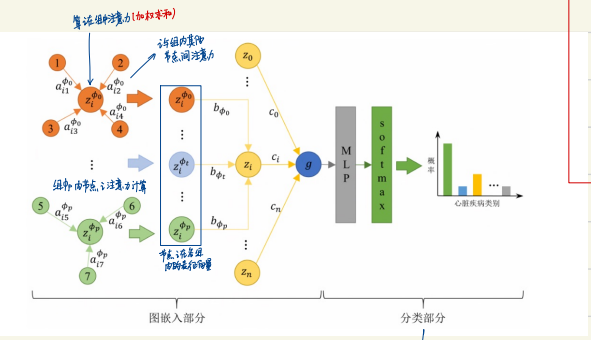
构建这样的网络

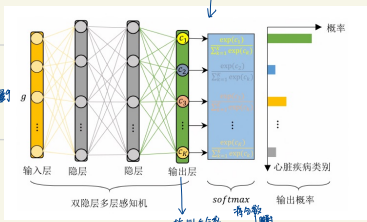
异构图分类任务

构建患者信息图

通过边类型分为多组

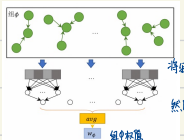
生成异构图数据集 —注意力机制→ 节点的表征向量 —聚合→ 节点的最终向量 —注意力机制→ 图嵌入向量 多层感知机 → 实现分类




计算各组权值

按照边的不同(颜色)，将中心结点和邻居节点划分成各组，



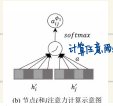

将组内结点映射为常数值

Leaky ReLu

然后求均值

$b_\phi$

计算各组内注意力




$e_{ij}^{\phi} = Leaky\ ReLU(a_\phi[h_i^\prime \| h_j^\prime])$

将拼接的隐层加权用作
映射力
度数 → Leaky ReLu 激活
softmax 归一化
得到 $\alpha_{ij}^{\phi}$ 在φ组内i和j的注意力系数

$z_i^\phi = Leaky\ ReLU\left(\sum_{j\in N_i^\phi} \alpha_{ij}^\phi \cdot h_j\right)$

加权求和，激活，得到组内中心的
最终向量
(注意力)

得到i在各组内的最终向量(注意力)
$\{z_i^1, z_i^2, z_i^3 \cdots z_i^{\phi_p}\}$

将i在各组内注意力加权求和

$$z_i = \sum_{t=1}^{p} b_{\phi_t} \cdot z_i^{\phi_t}$$

图i的新节点表征向量

计算各结点对整个图的注意力系数

(a) 计算图嵌入向量的加权求和

(b) 图注意力计算示意图

$g_i = Leaky\ ReLU(c[g_0 \| z_i])$

将注意力系数 加权求和，

最终得到 图嵌入向量g

$g = \sigma\left(\sum_{i\in V} c_i \cdot z_i\right)$

节点对图注意力系数

选择结点
用异质信息网络进行映射
度数
Leaky ReLu 激活

Unicode : U + 1组16进制数来表示字符

#decode('utf-8') 把 UTF-8 转化为 Unicode 编码

#encode('utf-8') 把 Unicode 转化为 UTF-8 编码

UTF-8 : 前 0x7F个字符与ASCII码对应

## 处理str

| 函数 | 功能描述 | 函数 | 功能描述 |
|---|---|---|---|
| capitalize() | 将字符串的第一个字符转换为大写 | max(str) | 返回字符串 str 中最大的字母 |
| count() | 返回 str 在 string 里面出现的次数 | min(str) | 返回字符串 str 中最小的字母 |
| find() | 检测 str 是否包含在字符串中 | rstrip() | 删除字符串末尾的空格 |
| index() | 跟 find()方法一样 | strip([chars]) | 在字符串上执行 lstrip()和 rstrip() |
| join(seq) | 以指定字符串作为分隔符，将 seq 中所有的元素(的字符串表示)合并为一个新的字符串 | swapcase() | 将字符串中大写转换为小写，小写转换为大写 |
| len(string) | 返回字符串长度 | upper() | 转换字符串中的小写字母为大写 |
| lower() | 转换字符串中所有大写字符为小写 | zfill (width) | 返回长度为 width 的字符串，原字符串右对齐，前面填充 0 |
| lstrip() | 截掉字符串左边的空格或指定字符 | isdecimal() | 检查字符串是否只包含十进制字符，如果是返回 true，否则返回 false |

## 读取 json 文件

- json.load()：将一个存储在文件中的json对象（str）转化为相对应的python对象
- json.loads()：将一个json对象（str）转化为相对应的python对象
- json.dump()：将python的对象转化为对应的json对象（str),并存放在文件中
- json.dumps()：将python的对象转化为对应的json对象（str）

一般用这个读，但是文件内一定要规范：

如 {"one":1,"two":2}

得是双引号，不然无法识别

```
import json
with open('data/train/haha.txt') as f:
    data = json.load(f)
    print(data['two'])
```

```
{"one": 1, "two": 2, "three": 3}
```

# 读写文件

① OS 库： os.path.exists() 判断是否存在路径

**os.path操作___按顺序创建文件夹**

os.listdir('.') ：返回列表，内容为该路径下的所有文件

```
num = len(os.listdir('data/preprocess'))
print(num)
if not os.path.exists('data/preprocess/preprocess_{}'.format(num)):
    os.makedirs('data/preprocess/preprocess_{}'.format(num))
```

num 返回文件夹个数

跟据地址创建文件夹（注意，只是文件夹，创建不了txt，CSV之类的。）

② panda 库读取 CSV，txt 文件。

1. 读 txt

```
import pandas as pd
stop_words = pd.read_table('data/stopwords/hahahah.txt',header = None,sep = ' ')
print(stop_words)

print('\n读取第一列')
stop_words = pd.read_table('data/stopwords/hahahah.txt',header = None,sep = ' ')
print(stop_words[0].tolist())
```

读取列名　　指定分隔符

→ 读第一列并且转化为列表

```
   0 1
0  a 1
1  b 2
2  c 3
3  d 4

读取第一列
['a', 'b', 'c', 'd']
```

← 给我加列名行名

txt 内容

hahahah.txt

文件　编辑　查看

```
a 1
b 2
c 3
d 4
```

CSV使用open读取

```
import codecs

with codecs.open('data/restaurant_comment/train.csv', 'r', 'utf-8') as f:
    for line in f.readlines():
        label, comment = line.strip().split('\t')
```

据说很高级！

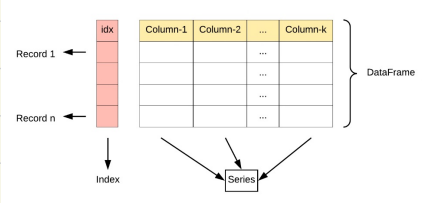
2. 读 CSV

```
import pandas as pd
df = pd.read_csv('data/restaurant_comment/train.csv',encoding= 'utf-8',sep = '\t')
print(df)
```

读取和comment数据集，一行多换行符，需用sep分隔

```
       label                                            comment
0          0                          一如既往地好吃，希望可以开到其他城市
1          0                                   味道很不错，分量足，客人很多，满意
2          0  下雨天来的，没有想象中那么火爆。环境非常干净，古色古香的，我自己也是个做服务行业的，我都觉得...
3          0                                 真心不好吃 基本上没得好多味道
4          0                少送一个牛肉饺堡 而且也不好吃 特别是鸡肉卷 **都不想评论了 谁买谁知道
...      ...                                                ...
```

| | A | B | C | D |
|---|---|---|---|---|
| 1 | labelcomment | | | |
| 2 | 0一如既往地好吃，希望可以开到其他城市 | | | |
| 3 | 0味道很不错，分量足，客人很多，满意 | | | |
| 4 | 0下雨天来的，没有想象中那么火爆。环境非常干 | | | |
| 5 | 0真心不好吃 基本上没得好多味道 | | | |
| 6 | 0少送一个牛肉汉堡 而且也不好吃 特别是鸡肉卷 | | | |
| 7 | 0用美团，图打折。 | | | |




3. 将 dataframe 保存入 CSV 文件

```
df.to_csv('data/preprocess/preprocess_1.csv',index=True,header=True)
```

不存在路径会自动创建

```
      Site  Age
No.1  Google  10
No.2  Runoob  12
No.3    Wiki  13
```

☆ 用with读取文件

```
with open('data/stopwords/哈工大停用词表.txt', encoding='utf-8') as f:  # 之前一直读不下来，结果是忘了加utf-8
    for word in f.read().split('\n'):
        print(word)
```

是遍历 f.read()

```
甚而
甚么
甚至
省得
时候
什么
什么样
```

# dataframe 基本操作

配合 lambda 函数使用

```
func = lambda x:x*1 if x==1 else 0
print(func(1))
print(func(2))

import pandas as pd
data = {'Site':['aa', 'bb', 'cc','dd'], 'p':[0.85, 0.92, 0.52,0.23]}
df_1 = pd.DataFrame(data)
print(df)
print("\n")
df_2 = df_1['p'].apply(lambda x:'{:.0%}'.format(x))
print(df_2)
```

传　　结果　　参数

→ 转为dataframe格式

选列名　　进行操作

```
2
0
  Site    p
0   aa  0.85
1   bb  0.92
2   cc  0.52
3   dd  0.23


0    85%
1    92%
2    52%
3    23%
Name: p, dtype: object
```

☆☆☆ csv 文件追加写
写在最后一行

```
import csv


# 如果不写 newline='', csv文件生成的时候，里面会每隔一行出现一个空白行
def write_csv():
    with open(r'write_down.csv', 'a+', newline='', encoding='utf8') as f:
        csv_write = csv.writer(f)
        data_row = ["1", "2"]
        csv_write.writerow(data_row)


write_csv()
```

如果使用绝对路径，使用字符串时需加r，如果写文件时序列'a'表示追加写，不加则覆盖原'的路径，不然需要换斜杠

在文件最后一行写。

☆☆　注意，运行时一定要关闭文件，否则会permission denied 报错

| | |
|---|---|
| w 只写 | 打开一个文件只用于写入。如果该文件已存在则打开文件，并从开头开始编辑，即原有内容会被删除。如果该文件不存在，创建新文件。 |
| r 只读 | 以只读方式打开文件。文件的指针将会放在文件的开头。这是默认模式。 |
| t | 文本模式 (默认)。 |
| x | 写模式，新建一个文件，如果该文件已存在则会报错。 |
| b | 二进制模式。 |
| + | 打开一个文件进行更新(可读可写)。 |
| a 追加写 | 打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。也就是说，新的内容将会被写入到已有内容之后。如果该文件不存在，创建新文件进行写入。 |

访问元组：同于访问数组

**实例(Python 3.0+)**

```
#!/usr/bin/python3

tup1 = ('Google', 'Runoob', 1997, 2000)
tup2 = (1, 2, 3, 4, 5, 6, 7 )

print ("tup1[0]: ", tup1[0])
print ("tup2[1:5]: ", tup2[1:5])
```

以上实例输出结果：

```
tup1[0]:  Google
tup2[1:5]:  (2, 3, 4, 5)
```

# Python 类操作

## ① 三种魔术方法

```python
class Fun:
    def __init__ (self, x_list):
        """ initialize the class instance
        Args:
            x_list: data with list type
        Returns:
            None
        """
        if not isinstance(x_list, list):
            raise ValueError("input x_list is not a list type")
        self.data = x_list
        print("intialize success")

    def __getitem__(self, idx):
        print("__getitem__ is called")
        return self.data[idx]

    def __len__ (self):
        print("__len__ is called")
        return len(self.data)

fun = Fun(x_list=[1, 2, 3, 4, 5])
print(fun[2])
print(len(fun))
```

初始化类的实例 (__init__ 魔术方法)

对象通过下标访问对象的数据集合，且触发"迭代"(__getitem__ 魔术方法)

求取对象的长度 (__len__ 魔术方法)

```
intialize success
__getitem__ is called
3
__len__ is called
5
```

## (创建对象) (通过实例) ②. 实例化，访问类属性和方法)

```python
class Myclass:
    i = 12345

    def f(self):
        return 'hello'


# 实例化类
x = Myclass()

# 访问类的属性
print(x.i)
# 访问类的方法
print(x.f())
```

```
12345
hello
```

## ③ 类的继承

```python
class people:
    name = ''
    age = 0
    __weight = 0

    def __init__(self, n, a, w):
        self.name = n
        self.age = a

    def showing(self):
        print('name:{},age:{}'.format(self.name, self.age))

    def show_age(self):
        print(self.age)


class student(people):
    grade = ''

    def __init__(self, n, a, w, g):
        super(people, self).__init__(n, a, w)  # super(后父类函数内不用再写self)
        # people.__init__(self, n, a, w)       # 调用父类的构造函数(于super()等价)
        self.grade = g

    def showing(self):
        people.showing(self)  # 调用父类的方法
        print('grade:{}'.format(self.grade))

    # 覆盖了父类的方法(直接写一个同名函数)
    def show_age(self):
        self.age = self.age - 5
        print(self.age)
haha = student('haha', 21, 66, 'undergraduate')
haha.showing()
haha.show_age()
```

继承父类属性的子类

可通过Super继承父类的方法

也可以用 父类.方法 来继承

覆写

## 类在项目中的应用

1. 数据集类继承自父类 Dataset，可用super来继承一些方法(不过这里没使用)

```python
class MinistDataset(Dataset):

    def __init__(self, img_path, mask_path, istrain: bool):

        def load_mnist_train():
```

```python
from torch.utils.data import DataLoader

# Dataloader的对象
train_loader = DataLoader(dataset=MinistDataset(train_img_path,
                                                train_mask_path,
                                                istrain=True),
                          batch_size=batch_size,
                          shuffle=False)
# TODO:SHUFFLE = FALSE这里把顺序去了

model = LLnet()  # 模型的对象
n_train = DataLoader.__len__(train_loader)  # 返回对象的长度
```

外部类

DataLoader(继承) 这个类，同时调用了新的类

创建一个对象

注意：返回的是 batch 的数量 !!!

对象通过 len 来访问自身长度

```python
class UpsamplingBlock(nn.Module):
    """
    Upsampling Block (upsampling layer) as specified by the paper
    This is composed of a 2d bilinear upsampling layer followed by a conv
    """

    def __init__(self, in_channels: int, out_channels: int, size: Tuple):
        """
        Create the layers for the upsampling block

        :param in_channels:   number of input features to the block
        :param out_channels:  number of output features from this entire
        :param scale_factor:  tuple to determine how to scale the dimensi
        :param residual:      residual from the opposite dense block to u
        """
        super().__init__()
        # bilinear vs trilinear kernel size and padding
        if size[0] == 2:
            d_kernel_size = 3
            d_padding = 1
        else:
            d_kernel_size = 1
            d_padding = 0

        self.upsample = nn.Upsample(
            scale_factor=size, mode="trilinear", align_corners=True
        )
        self.conv = nn.Sequential(
            nn.Conv3d(
                in_channels,
                out_channels,
```

从 nn.Module 继承的 upsample 类

train_loader

__len__ 方法：返回对象的 batch 数目

```
# Dataloader的对象
train_loader = DataLoader(dataset=MinistDataset(train_img_path,
                                                train_mask_path),
                          batch_size=batch_size,
                          shuffle=True)

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model = LLnet().to(device)  # 模型的对象
num_of_batch = DataLoader.__len__(train_loader)  # todo:返回对象的长度(注意，这里是num/batch_size=batch)返回的是batch的数量
n_train = num_of_batch * batch_size
```

调用模型的__init__()赋值，主体 output = model(inputs) 调用的是 nn.Module 的 forward 方法！

# numpy高维数组切片

```python
import numpy as np
arr = np.array([[1,2,3,5],[1,2,3,6]],[[1,2,3,5],[1,7,8,9]],[[2,4,5,6],[6,7,8,9]]])
print(arr)
print()
print(arr.shape)

# 高维切片操作
print()
print(arr[1:2,1:2,1:2])          # 表示过滤一维度都取索引1到索引2的元素
print(arr[1:2,1:2,1:2].shape)


print()
print(arr[1,:,:])
print(arr[1,:,:].shape)          # 截取第一个维度


print()
print(arr[1,:,:])
print(arr[:,:,1].shape)      # 截取第三个维度
```

```
[[[1 2 3 5]
  [1 2 3 6]]

 [[1 2 3 5]
  [1 7 8 9]]

 [[2 4 5 6]
  [6 7 8 9]]]

(3, 2, 4)
```

arr

arr.shape

```
[[[7]]]
(1, 1, 1)

[[1 2 3 5]
 [1 7 8 9]]
(2, 4)

[[1 2 3 5]
 [1 7 8 9]]
(3, 2)
```

像[::2]这种的 是外

```python
import numpy as np
arr = np.array([1,2,3,4,5,6,7,8,9])
print(arr[::3])  #最后的数字代表隔开的步长
```

```
[1 4 7]
```

# np遍历featurmap

```python
import numpy as np
arr = np.zeros((3, 255, 255))  # (C,H,W)
for i in range(arr.shape[0]):
    print(arr[1, :, :])
```

https://www.runoob.com/numpy/numpy-array-manipulation.html

# numpy基操

## ①np.reshape

**numpy reshape函数**

不改变值只改变形状

reshape(-1,1) 转化成一列

reshape(1,-1) 转化成一行

reshape(2,-1) 转化成两行

```
原始数组：
[0 1 2 3 4 5 6 7]

修改后的数组：
[[0 1]
 [2 3]
 [4 5]
 [6 7]]
(4, 2)
```

```python
import numpy as np

a = np.arange(8)
print ('原始数组：')
print (a)
print ('\n')

b = a.reshape(-1,2)
print ('修改后的数组：')
print (b)
print(b.shape)
```

## ②np.squeeze

**numpy squeeze函数**

从数组的形状中删除一维条目

```python
import numpy as np
a = np.arange(2).reshape(1,1,2)
print(a)
print(a.shape)
b = np.squeeze(a)
print(b)
```

```
[[[0 1]]]
(1, 1, 2)
[0 1]
```

## ③np.transpose

**numpy transpose函数**

对换数组的维度

np.transpose(arr,[2,1,0]) 第二个参数为一个列表

```python
import numpy as np

a = np.arange(24).reshape(2,3,4)

print ('原数组：')
print (a)
print ('\n')

print ('对换数组：')
print (np.transpose(a,[1,0,2]))
```

```
原数组：
[[[ 0  1  2  3]
  [ 4  5  6  7]
  [ 8  9 10 11]]

 [[12 13 14 15]
  [16 17 18 19]
  [20 21 22 23]]]

对换数组：
[[[ 0  1  2  3]
  [12 13 14 15]]

 [[ 4  5  6  7]
  [16 17 18 19]]

 [[ 8  9 10 11]
  [20 21 22 23]]]
```

数据维数：

三维：

三维灰度图

四维

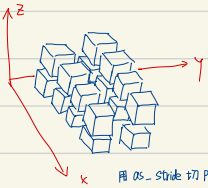
1维+3维

D[i] → 3维矩阵

六维

3维+3维

D[i][j][k] → 3维

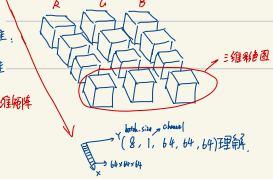



用 OS_Stride 切 patch 后切出来的是6维数据

五维：

2维+3维

D[i][j] → 3维矩阵



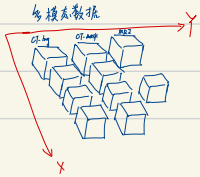

三维彩色图

batch_size → channel

(8, 1, 64, 64, 64)理解

64×64×64

多模态数据



# numpy 增删操作

## ①. np.append（自动合并）

```python
import numpy as np
aaa = np.arange(0, 6).reshape(2, 3)
bbb = np.arange(8, 14).reshape(2, 3)
ccc = np.append(aaa, values=bbb)    # 不指名行和列就会拉成一排
ddd = np.append(aaa, values=bbb, axis=0)    # 按行拼接
eee = np.append(aaa, values=bbb, axis=1)    # 按列拼接
```

按行
按列

```
G:\ANACONDA\python.exe E:\savefryomxftp\7.31MON_NLP\temp.py
[ 0  1  2  3  4  5  8  9 10 11 12 13]

[[ 0  1  2]
 [ 3  4  5]
 [ 8  9 10]
 [11 12 13]]

[[ 0  1  2  8  9 10]
 [ 3  4  5 11 12 13]]
```

## ②. np.insert（自动合并）

```python
import numpy as np
aaa = np.arange(0, 6).reshape(2, 3)
bbb = np.arange(8, 14).reshape(2, 3)
ccc = np.zeros(shape=(1, 1))
ddd = np.insert(aaa, obj=1, values=ccc, axis=0)
eee = np.insert(aaa, obj=2, values=ccc, axis=1)
```

插入行，第1行
插入列，第2列

```
[[0.]]

[[0 1 2]
 [0 0 0]
 [3 4 5]]

[[0 1 0 2]
 [3 4 0 5]]
```

## ③. np.delete

```python
import numpy as np
aaa = np.arange(0, 6).reshape(2, 3)
bbb = np.arange(8, 14).reshape(2, 3)
ccc = np.zeros(shape=(1, 1))
ddd = np.delete(aaa, 1, axis= 0)    # 删除行,第二行
eee = np.delete(aaa, 1, axis= 1)    # 删除列,第二列
```

```
[[0 1 2]]

[[0 2]
 [3 5]]
```

## ④. 不需要自动合并

```python
import numpy as np    拼表list → 用append → 转成array

aaa = np.arange(0, 8).reshape(2, 2, 2)
aaa = aaa.tolist()
bbb = np.arange(8, 16).reshape(2, 2, 2)
bbb = bbb.tolist()

ccc = list()
ccc.append(aaa)
ccc.append(ccc)

转化为array !!
ddd = np.array(ccc, dtype=object)
print(ddd.shape)
```

```
(2, 2, 2, 2)
```

而对于列表list，却**不能使用shape**来查看列表的维度。 如果非要想查看列表维度，可以采用： T1、将列表转为array格式，然后使用shape即可！

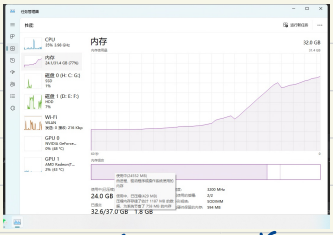
# numpy 循环切片导致内存积累溢出解决办法！

```python
import numpy as np
n = 1000
l = []
for i in range(n):
    array_large = np.random.choice(1000, size=(7000, 7000))
    array_small = array_large[:5, :5].copy()
    l.append(array_small)
```

↑ 解决该问题

这种切片操作实际上是视图
视图指向原图，就算 del array_large
也删不掉

因此在 array_large[:5, :5].copy()
使 array_large 真正赋值给了 array_small
使其最终能被释放掉




内存随 n 线性增涨
n 大到一定程度程序崩溃

↓

原因：

```python
def load_image(image_path, x, y):
    image = np.load(image_path)
    image = image[x:x+32, y:y+32, :]
    if np.random.random() > 0.5:
        image = np.flip(image)
    return image

images = []
for line in lines:
    image_path, x, y = line.split(' ')
    x, y = int(x), int(y)
    image = load_image(image_path, x, y)
    images.append(image)
np.save('images/images.npy', images)
```

像这种如果给 1000 张图到 load_img 做切片操作，
原图 img 的内存不会被释放，累积下来
最终程序要炸。

patch 切的实在太多
都指向同一张原图，内存炸了 是因为

ps. 但是像 spine_segmentation 中切 3D patch 与上述无关!!!，同一张图产生的 patch

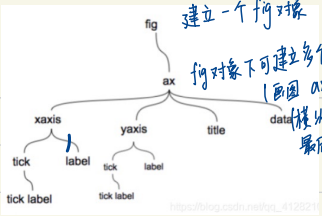
---

np.all (ndarray, axis)　判断该轴是否所有元素均为 True

np.any (ndarray, axis)　判断 . . . 是否有一个 True

判空：　ndarray.max = 0.

# matplotlib库解析




(图像对象，称为大画布)

建立一个fig对象 (fig = plt.figure())

fig对象下可建立多个ax对象

(每个图像的内容对象，如一张图输出八条曲线)

(画图 ax.plot())

(设坐标：ax.xaxis())

最后输出：plt.show()(整个

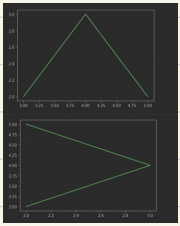
```
import matplotlib.pyplot as plt

fig_1 = plt.figure()
fig_2 = plt.figure()

axes_1 = fig_1.add_axes([0.1,0.1,0.8,0.8])
axes_2 = fig_2.add_axes([0.1,0.1,0.8,0.8])

x=[3,4,5] # [列表]
y=[2,3,2] # x,y元素个数N应相同
axes_1.plot(x,y,'g')
axes_2.plot(y,x,'g')
plt.show()
```

建立两个fig对象

新fig对象建立一个axes对象并设置初值

对象.plot(横坐标,纵坐标,颜色)

最后总的输出(两个fig)



```
import matplotlib.pyplot as plt
from pylab import *
x = linspace(0, 5, 10)
y = x ** 2

fig = plt.figure()

axes = fig.add_axes([0.1, 0.1, 0.8, 0.8])# left, bottom, width, height (range 0 to 1) 前两个参数表示在figure中的位置
axes2 = fig.add_axes([0.2, 0.5, 0.4, 0.3])# inset axes

axes.plot(x, y, 'r')#参数3 为设置颜色

axes.set_xlabel('x')
axes.set_ylabel('y')
axes.set_title('title')

# insert
axes2.plot(y, x, 'g')
axes2.set_xlabel('y')
axes2.set_ylabel('x')
axes2.set_title('insert title')

plt.show()
```

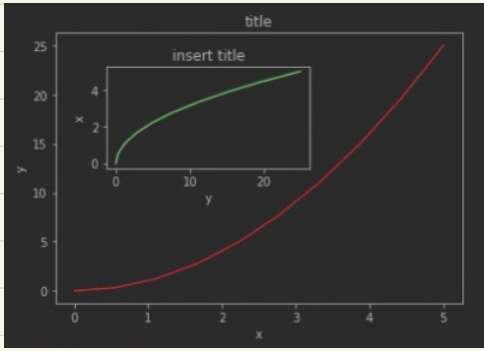
建立一个图像对象

一个图像对象的两个axes对象

后6-前面中

axes

axes2




去除坐标轴：plt.axis('off')

plt读取和保存RGB图像 ⇒ 直接读成numpy

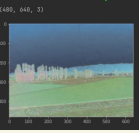
```
import matplotlib.pyplot as plt
import os
num = len(os.listdir('result/featuremap'))

img = plt.imread('data/image/114514.jpg', format=None)   # format读取文件格式，None为自动读取，读下来是numpy形式
print(img.shape)
plt.imshow(img)
plt.imsave('result/featuremap/'+str(num)+'.png', img)
plt.show()
```

'hot' 热力图

plt.imshow(img, cmap='gray')

灰度图

```
(480, 640, 3)
```



## Show_featuremap

```
fig = plt.figure()
```

→ 建立对象

```
ax_0 = fig.add_subplot(2, 3, 1)
ax_0.set_title('')
plt.imshow(slice_0, cmap="gray")
ax_1 = fig.add_subplot(2, 3, 2)
ax_1.set_title('')
plt.imshow(slice_1, cmap="gray")
ax_2 = fig.add_subplot(2, 3, 3)
ax_2.set_title('')
```

子图对象

子图标题

灰度图

总标题

```
plt.suptitle("ID={}\t\tDC={}".format(str(img_ID), str(DC)))
```

```
plt.savefig('result/910/featuremap/ID{}.png'.format(str(img_ID)), dpi=600)
```

保存

分辨率

# Linux 指令

1. sudo：获取权限 root ubuntu

2. wget，下载软件

3. 杀死前台进程

如果命令/进程在前台运行，您可以使用 Ctrl+C 终端快捷方式。但是，如果进程不可见（在后台运行），您可以使用专用命令“杀死它”。

“终止进程”是指在执行过程中停止进程，如果您知道进程 ID (PID)，则可以使用 kill 命令，如下所示：

```
kill <signal> <PID>
```

**第一、安装环境编译包**

```
sudo apt update
sudo apt install build-essential zlib1g-dev libncurses5-dev libgdbm-dev libnss3-dev li
```

**第二、下载最新python软件包**

```
cd /opt
sudo wget https://www.python.org/ftp/python/3.9.0/Python-3.9.0.tgz
```

这里我们下载最新的3.9包版本。

```
tar xzf Python-3.9.0.tgz
cd Python-3.9.0
sudo ./configure --enable-optimizations
```

① 打开D盘

```
C:\Users>d:

D:\>
```

1. apt指令.

apt

## 第一、安装环境编译包

获取管理员权限

```
sudo apt update
sudo apt install build-essential zlib1g-dev libncurses5-dev libgdbm-dev libnss3-dev li
```

检查更新

## 第二、下载最新python软件包

相当于在浏览器输入这一段网址后下载一个安装包

```
sudo wget https://www.python.org/ftp/python/3.9.0/Python-3.9.0.tgz
```

很快

这里我们下载最新的3.9包版本。

要加一个sudo

```
tar xzf Python-3.9.0.tgz
cd Python-3.9.0
sudo ./configure --enable-optimizations
```

## 第三、安装详细过程

```
sudo make altinstall
```

这里我们安装，然后可以就去看看版本是不是python3.9.

当然，上面是我们自己直接使用的Python官方的安装包，我们也可以直接在UbUNTU上面直接编译。

```
sudo apt update
sudo apt install software-properties-common
```

安装需要的源，然后我们添加到列表包。

```
sudo add-apt-repository ppa:deadsnakes/ppa
```

在过程中我们需要回车。

然后我们直接安装

```
sudo apt install python3.9
```

可以只要这一行，腾讯云用自己的镜像源下载

执行setup.py文件下载

装在opt里的

'/' → home → ubuntu → Spine_segmentation

opt → python3.9

进下一个文件：cd

回退：cd ../

然后我们检查版本。

然后我们检查版本。

```
python3.9 --version
```

在python 3.9这个编译环境下换python版本

## ②切换python编译环境

```
sudo update-alternatives --list python
```

列出可替换的python版本

如果报错 → 则更新列表

现在就有两个环境了：python 3.8和python3.9

python train.py → 用的python 3.8环境

python 3.9 train.py → 用的python 3.9环境

要用python 3.9环境

就写python 3.9

用python 3.8环境

就写python

# ls 操作

ls 显示当前文件夹下文件

ls -l 列表显示

ls -a 显示隐藏文件

ls -h 文件大小，自动找单位（Kb、M、G）

也可以 ls -l -a -h (-l -a -h没有先后顺序，-a -h -l)

也可以 ls -lah (-lah没有先后顺序，-ahl)

ls加点正则表达

ls a* 显示以a开头

ls *a 显示以a结尾

ls a? 显示以a开头，后面仅1位

ls a?? 显示以a开头，后面仅2位

ls 1[12345]4 显示以1开头,中间一位12345任意，最后一位4

直接显示文件

```
ubuntu@VM-0-10-ubuntu:~$ ls spine_segmentation
 config.yaml   dataset      evaluation   model                __pycache__   temp.txt   training          'Validation Coeff.jpg'
 data          evaluate.py  main.py      prepare_dataset.py   result        test.out   'Train loss.jpg'
```

列表显示并且标明文件大小

```
ubuntu@VM-0-10-ubuntu:~$ ls -l -h spine_segmentation
total 672K
-rw-rw-r-- 1 ubuntu ubuntu 1.2K Aug  4 10:45  config.yaml
drwxrwxr-x 6 ubuntu ubuntu 4.0K Aug  8 15:33  data
drwxrwxr-x 4 ubuntu ubuntu 4.0K Aug  4 16:35  dataset
-rw-rw-r-- 1 ubuntu ubuntu 3.3K Aug  4 16:35  evaluate.py
drwxrwxr-x 3 ubuntu ubuntu 4.0K Aug  4 16:35  evaluation
-rw-rw-r-- 1 ubuntu ubuntu 1.2K Aug  4 16:35  main.py
drwxrwxr-x 4 ubuntu ubuntu 4.0K Aug  4 16:35  model
-rw-rw-r-- 1 ubuntu ubuntu 2.7K Aug  4 16:35  prepare_dataset.py
drwxrwxr-x 2 ubuntu ubuntu 4.0K Aug  4 17:54  __pycache__
drwxrwxr-x 8 ubuntu ubuntu 4.0K Aug  8 18:17  result
-rw-rw-r-- 1 ubuntu ubuntu 1.9K Aug  4 17:36  temp.txt
-rw-rw-r-- 1 ubuntu ubuntu 573K Aug  9 00:08  test.out
drwxrwxr-x 4 ubuntu ubuntu 4.0K Aug  4 17:36  training
-rw-rw-r-- 1 ubuntu ubuntu  19K Aug  9 00:04 'Train loss.jpg'
-rw-rw-r-- 1 ubuntu ubuntu  22K Aug  9 00:04 'Validation Coeff.jpg'
```

在 ubuntu 中查找 python 的位置

在 /usr/bin 里面，在 cd / 再 cd usr cd bin 进入

```
ubuntu@VM-0-10-ubuntu:~$ whereis python
python: /usr/bin/python2.7 /usr/bin/python3.6m /usr/bin/python /usr/bin/python3.6 /usr/bin/python3.6-config /usr/bin/python3.6m-config /usr/bin/python3.7 /usr/bin/python
3.7m /usr/bin/python3.8 /usr/lib/python2.7 /usr/lib/python3.6 /usr/lib/python3.7 /usr/lib/python3.8 /etc/python2.7 /etc/python /etc/python3.6 /etc/python3.7 /etc/python3
.8 /usr/local/bin/python3.9 /usr/local/bin/python3.9-config /usr/local/lib/python2.7 /usr/local/lib/python3.9 /usr/local/lib/python3.6 /usr/local/lib/python3.7 /usr/loca
l/lib/python3.8 /usr/include/python3.6m /usr/include/python3.6 /usr/share/python /usr/local/miniconda3/bin/python /usr/local/miniconda3/bin/python3.8 /usr/local/minicond
a3/bin/python3.8-config /usr/share/man/man1/python.1.gz
```

# windows 切换环境

在base这个环境中

```
(base) PS E:\savefryomxftp>  conda env list
# conda environments:
#
base                  *  G:\ANACONDA
                         G:\ANACONDA\Scripts
good                     G:\ANACONDA\envs\good
nnUnet_win               G:\ANACONDA\envs\nnUnet_win

(base) PS E:\savefryomxftp> ^A
```

之前因为powershell禁止运行脚本，但anaconda后再安装powershell要运行脚本所以报错，加了挂久。

| 功能 | 命令 |
|---|---|
| 创建虚拟环境 | `conda create -n 环境名称 python=3.6` (python版本为3.6) |
| 删除虚拟环境 | `conda remove -n 环境名称 --all` |
| 激活虚拟环境 | `conda activate 环境名称` |
| 关闭虚拟环境 | `conda deactivate` |
| 查看已安装的库 | `conda list` |
| 列出当前已有环境 | `conda env list` |
| 删除环境中某个库 | `conda remove 包名称` (已进入虚拟环境内) |
| 查看已存在虚拟环境 | `conda env list` |
| 安装库 | `pip install 名称 -i https://pypi.tuna.tsinghua.edu.cn/simple` (使用镜像，下载更快) |
| 安装指定版本的库 | `pip install 名称==版本 --user` (如，`pip install pillow==6.2.2 --user`) |
| 打开jupyter book | `jupyter notebook` |
| 添加Anaconda的TUNA镜像 | `conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/free/` |

```
conda create -n nnUnet_win python=3.7
```

创建虚拟环境

自命名称

```
conda activate nnUnet_win
```

激活虚拟环境
(切换虚拟环境)

conda 报错解决办法

rm ~/.condarc

**查看cuda，输入：**

```
nvcc -V
```

在pytorch官网上找对应cuda的pytorch版本 (没装过的话了了一放了装下来网)

# 查看该环境的pytorch版本

先输入python进入编译状态，再 -version 即可查看

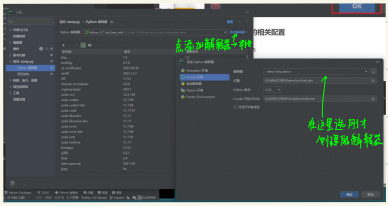
```
(nnUnet_win) PS E:\savefryomxftp\nnUnet_win\nnUNet-Windows> python
Python 3.7.16 (default, Jan 17 2023, 16:06:28) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import torch
>>> torch.__version__
'1.13.1'
>>> import torchvision
>>> torchvision.__version__
'0.14.1'
```

查看pytorch cuda是否可以使用 (exit()退出)

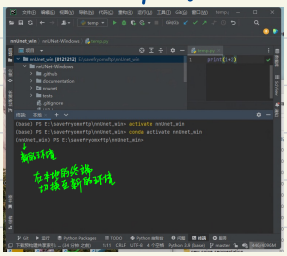
```
>>> x = torch.rand(5, 3)
>>> print(x)
tensor([[0.1704, 0.9490, 0.0827],
        [0.2161, 0.2408, 0.2223],
        [0.8135, 0.7266, 0.7247],
        [0.1498, 0.8910, 0.7205],
        [0.3023, 0.3644, 0.6441]])
>>> torch.cuda.is_available()
True
>>>
```

pip install -e ⟶ 会执行当前目录下的setup.py文件

配置pycharm虚拟环境




在本地终端也可切至新环境



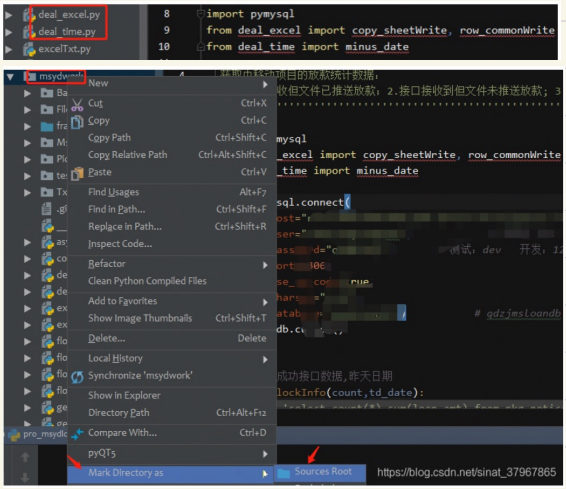

完成

# Linux 切换环境

① 获得root权限

```
ubuntu@VM-0-10-ubuntu:~$ su -
Password:
root@VM-0-10-ubuntu:~#
```

pycharm 路径报错：→ 将目录标记为资源根




运行windows nnUnet 时缺少 batchgenerators 这个包

pip install batchgenerators==0.21

# 一、数据格式转换

.nii .raw .dcm ⇒ { numpy 四维数组 (可看成多个三维图像)
pickle spacing direction origin 信息.

[-1,:,:,:]

MRI { [0,:,:,:] 强度.
[1,:,:,:] T1 加权强度.

第一维度最后是不同模态、标注，第一维度前面是不同模态、数据
如果是单模态，则第一维为2，为{ 单张数据 标注数据
0,1,2代表不同标注类别

[[模态一][模态二][标注数据]]
4维的 data (CXYZ) 3维的 seg (XYZ)

# 二. 图像裁剪 crop

在原始图像 data 中寻找最小三维区域，将其裁剪为0
并顺便裁剪 seg

# 采样及patch方法

采样 Sample 将图片分成 $n \times m$ 个像素(pixel)，连续 ⇒ 离散

↓

即图像分辨率为 $n \times m$ (像素个数)(尺寸)

下采样：downsample　分辨率减小

上采样：upsample　分辨率提高

像素间距：spacing 体素　某个像素代表的真实长度

图像真实大小：分辨率 × 体素

由于不同设备采样会产生不同体素的数据集，CNN无法理解体素间距

⇓

因此需要将医学影像重采样 resample 到一致

但是重采样后像素个数又不一样了.(尺寸)

无法放入模型训练

⇓

用patch (假设图片大小为 a×b pixel×b pixel)

- 上采样 (pixel：↑个数)　获得上下文信息↓，细节信息↑
- 下采样 (pixel：↓个数)　获得上下文信息↑，细节信息↓

MaxPooling

为什么使用patch？

放入模型必须resize成同一尺寸

需要确保体素一致，又要重采样导致尺寸不同

- 数据收集到的图像大小不一致，无法统一进行训练，如果强行Resize，会对图像产生扭曲，缩放，可能会丢失信息
- 图像规模非常大，显存不够
- 即使在显存充足的情况下，对于大图，模型可能难以捕捉到其中的细节，尤其对于动辄几个G的医学图像，可能绝对只有那么几十个像素的大小
- 数据集太小，数据难以收集。或者大量数据被污染，只有少量数据包含了明显的特征。
- .....

→ 将一整张图丢进去训练

可能又有算一点要求就学不到

而要manipulating 工作都学不到了.

patch切分方法：要保证相邻的patch间有一定重合

可以 stride = $\frac{patch}{2}$

原因：模型处理有边缘效应(边上处理不好) patch = stride 会有遗漏

patch切的大小

- 切大：不好凑整数个patch导致边上大量信息遗失
- 切小：学不了多少

坏处

好处

无法学到更多周围的关系

之前我们利用2D全卷积神经网络的分割算法实现对脑肿瘤的图像分割，尽管改进后的算法有很大的提升，但是分割依然存在一个问题，MRI脑肿瘤本身是一个三维数据结构，精确的3D模型对于医生指导诊断治疗有重要意义，而基于2D网络的分割算法需要将其进行切片，切片后的数据为2D数据，这使得算法无法学习到数据的层与层间的结构关系，导致网络模型学习不佳。并且，后期单纯对2D切片分割结果堆叠可能会形成斑点或断层现象，缺失了肿瘤的空间特征表达。因此，利用3D全卷积神经网络来解决这类问题是非常有必要的。

因此不要转换到2D求取patch

很灵活

# patch初步操作 (二维图像)

```python
import os
import matplotlib.pyplot as plt
import numpy as np
from patchify import patchify, unpatchify
patch_size = 64
patch_step = 32
```

→ 有一部分是重叠的，避免边缘效应

{372×372×3}

numpy读入是 (H,W,3)，断点看看时就是 H,C,W，不方便

还有通道

```
img = plt.imread("data/image/0.png", format=None)
img = np.transpose(img, [2, 0, 1])
```

转化为(C,H,W)

注意是切成 (3×64×64)

```
patches = patchify(img, (3, 64, 64), step=32)  # split image into 2*3 small 2*2 patches.
for i in range(patches.shape[1]):
    for j in range(patches.shape[2]):
        patch = patches[:, i, j, :, :]
        patch = np.squeeze(patch)
        patch = np.transpose(patch, [1, 2, 0])
        patchimg_num = len(os.listdir('result/featuremap'))
        plt.imsave('result/featuremap/' + str(patchimg_num) + '.png', patch)
```

遍历，保存

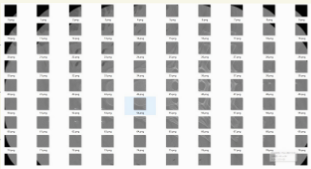
{3×64×64}

```
patch_num = int((img.shape[1]-patch_size)/patch_step) + 1
new_img_size = (patch_num-1)*patch_step+patch_size
```

→ 计算这张原图的图像大小
因为图像卷积实际分出patch所以但边有缺失
↓有需加回去的部分在此处.

```
reconstructed_image = unpatchify(patches, (3, new_img_size, new_img_size))
reconstructed_image = np.transpose(reconstructed_image, [1, 2, 0])
patchimg_num = len(os.listdir('result/featuremap'))
plt.imsave('result/featuremap/' + str(patchimg_num) + '.png', reconstructed_image)
```

{3×352×352}



切割结果
有一部分重叠.

切成 32×32×3 的切片

切之后为 {22, 22, 1, 32, 32, 3}
H W C H W C

☆ patch的步数计算方法

$$\frac{(\text{原图边长} - \overset{\text{patch_size}}{\text{卷积核边长}}) + 2\ \text{padding}}{\underset{\text{patch_step}}{\text{步长}}} + 1 = \text{步数}$$

原图：372×372
patch_size = 64
patch_step = 32

$$\text{patch_num} = \frac{372-64}{32} + 1 = 10.625$$

计算新的图的边长
取整：10
(10−1) × 32 + 64
= 352

patch策略. resample到同spacing (Resample) ⇒ 大小不一致 ⇒ patch方法解决
⇑
切割掉黑框 corp

首先，由于数据的稀少，标注成本高，医疗图像几乎必须要做数据增光。

医疗图像不同机器拍的spacing都不一样，不同于自然图像，每一个像素点代表的实际大小是不一样的。因此需要使用类似sitk把他们resampling到一样的大小，后导出numpy array。

而后由于图像太大，一般需要导出corp出一定的patch，一般采取随机corp，之后随机的进行数据增广，比如旋转，翻转，弹性形变这样的。

但是由于弹性形变等太花时间和CPU资源，我开发了一套在GPU上进行数据增广的代码，项目地址 github.com/qsyao/cuda_s...欢迎关注！

编辑于 2019-07-13 11:02

应该是**前景和背景像素的不平衡**吧

前景信息：脊柱，病灶 这种要预测的东西.

在医学图像分割模型，最常见的是基于分布的损失函数，如交叉熵损失函数 或CE Loss，在国际医学图像计算和计算机辅助干预协会于2018年举办的竞赛进程中，77 篇基于卷积神经网络的分割论文中有 47 篇选择了CE Loss 运用在分割模型中。 CE Loss是对所有像素的预测进行评估，但是当数 据集极度不平衡时，容易使模型陷入局部最优解， 从而使预测值极易偏向背景。例如，当90%的像素属于背景时，模型只需简单地将每个像素预测为背景即可实现 90% 的整体分类准确率，但这显然是不符合实际情况的。

全为背景的patch太多导致模型全预测为背景

→ 之前看到报告是说训练时只训前景信息.(脊柱部分)
↓
剩下的背景随便丢几张这么搞
(?)找不到出处了.

模型具有泛化能力

我之前做过的一个高分遥感影像语意分割的项目仿佛和你描述的场景有几分相似，所以来答一波。

用的GF2影像，一幅影像边长超过一万八千像素，图像文件大小超过10G，别说一次调入显存了，当时的计算机一次读入内存都费劲。所以采用了分块处理机制，一次处理1024*1024的patch，然后再读取下一个...但是由于patch在接边处会有对象割裂这种情况，所以相近patch之间有重叠。我一般设定的是512或者256的重叠。在重贴区域赋予类别时，取重叠两次或多次预测结果中得分最高的那个类别。举个例子，假如设定的patch重叠区域为256像素，那意味着这256像素的区域被两个patch处理过，针对其中某个像素，第一个patch以0.8的最大概率预测为水体，第二个patch以0.9的最大概率预测为草地，那就赋予后者类别，每个重叠区域的像素都按照这个方式进行处理。以上处理方式意味着你得以某种方式记录重叠区域的多次预测结果，不是只记录类别，而是得记录得分。

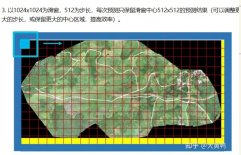
**最终预测**：采用交叠滑窗(滑窗步长<滑窗窗口大小)，预测时，只保留预测结果的中心区域，舍弃预测不准的周围边缘。

具体实现：

1. 填充1（黑色部分）：填充右下边界至滑窗按照窗口大小向整数倍，方便整除切割；

2. 填充2（蓝色部分）：填充1/2滑窗步长大小的外边框（为也边缘数据提供新预测）；

3. 以1024x1024为滑窗，512为步长，每次预测后冷保留滑窗中心512x512区域预测结果（可以满足最大部分长，和滑窗步长的中心区域，提高效率）。



# 用numpy切3D patch详解，(虽然很简单，但是有点绕)

```
for i in range(0, itk_img.shape[0] - self.patch_size[0], self.patch_stride[0]):
    for j in range(0, itk_img.shape[1] - self.patch_size[1], self.patch_stride[1]):
        for k in range(0,  itk_img.shape[2] - self.patch_size[2], self.patch_stride[2]):
            img_patch = itk_img[i:i + self.patch_size[0], j:j + self.patch_size[1], k:k + self.patch_size[2]]
            mask_patch = itk_mask[i:i + self.patch_size[0], j:j + self.patch_size[1], k:k + self.patch_size[2]]
            self.preprocessed.append(tuple([img_patch, mask_patch]))
```

公式：$\dfrac{\text{原图边长} - \text{卷积核边长} + \text{步长}}{\text{步长}} = \text{步数}$

两边同时乘以步长 即是每一步的落脚点。

但由于步数的计数是从0开始。 (0, 1, 2) 是3步，但是最大范围是2

因此要减去 patch-stride

即：for i in range(0, img-patch-size + patch-stride, patch-stride)

少加一个 patch-stride 就对了！

最终结果：for i in range(0, img-patch-size, patch-stride)

# np.percentile() 计算百分位数

**百分位是一个多维数组中对应每个数的，小于它的数的占总数比。**

```
np.percentile(a, q, axis=None,keepdims:bool)
```

被计算的对象　轴　是否保持维数不变

几分位数　=0.50 百分位.75

**参数**

- a : array，用来算分位数的对象，可以是多维的数组
- q : 介于0-100的float，用来计算是几分位的参数，如四分之一位就是25，如要算两个位置的数就[0,100]，多个位置如[0,25,50,75,100]
- axis : axis为0，在纵列上求;axis为1，在横列上求;
- keepdims:bool,是否保持维度不变。

```
import numpy as np
a = [[2,3,6],[10,11,12]]
print(np.percentile(a,[0,25,50,75,100]))
print(np.percentile(a,[0,50,100],axis =0))
```

最小数　上　中位数　下　最大数

这个位置的数是一行求取分位数 如果是两个则取两个数

axis=0 横列上求 axis=1 纵行上求

```
[ 2.    3.75  8.   10.75 12.  ]
[[ 2.  3.  6.]
 [ 6.  7.  9.]
 [10. 11. 12.]]
```

## 按位与

注意：都为1时才为1

### 按位与

```
a = 3   # 0011
b = 5   # 0101
print(a & b)    # 0001
```

```
1
```

## 25%数计算方法

排序为2，3,6,10,11,12，首先占比为0和占比100的当然就是最小值和最大值了；其次占比为25%的数小于3.75；再次占比为50%的数小于8，即小于中位数；最后占比为75%的数小于10.75；

为什么是3.75和10.75？下面介绍线性插值法计算百分位数

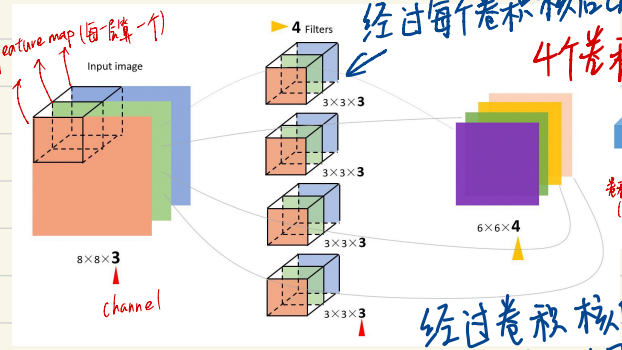
loc = 1 + (n - 1) * p，其中n为排序后数的个数，p为百分位

num = a[loc整数部分 - 1] + (a[loc整数部分] - a[loc整数部分 - 1]) * loc小数部分

复制

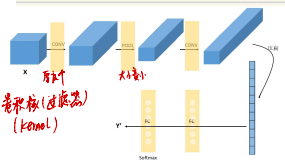
```
#Nlo = [2,3,6,10,11,12]
#对于25%
loc = 1+(6-1)*0.25 = 2.25
num = a[2 - 1] + (a[2] - a[2 - 1])*0.25
    = a[1] + (a[2] - a[1])*0.25
    = 3 + (6-3)*0.25
    = 3.75
```

# 对 Unet 的重新理解



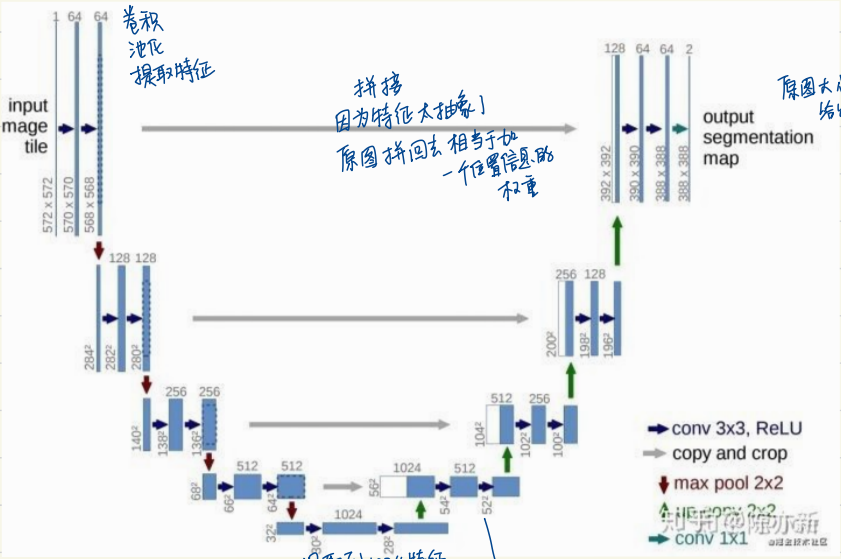

经过每个卷积核后 channal 变为 1

4个卷积后的结果叠在一起

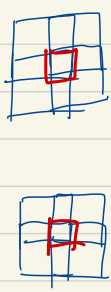



经过卷积核后
最后得到的是一个数

n个卷积核得到n个
feature map



# 由patch拼接回原来大小




一个像素点会被重复计算8次

⇓

乘一个高斯权重，

再求均值得到最终概率

①. 先跑一个无重叠的观查，

↳ 调参 { 高斯权重σ
patch size
patch step

需要为每个文件建立一个文档 ⟶ 图片尺寸
原图编号.

forward 函数：来自 nn.Module 的函数

☆注意：model.to(device) 是 nn.Module 里的
因此想将 python 部署在 GPU 上建议建立一个类来继承 nn.Module.

```
class net (nn.Module)
    def __inite__(self):
        super__inite__()
```

在创建网络的对象时调用.

部署在GPU上.

model = net().to(device)

```
def forward (img:tensor):
```

在调用对象时调用

output = model(input_img)

苟全和

# 其他.

- **随机过采样**：在少数类中随机复制示例。
- **随机欠采样**：随机删除多数类中的示例。

自动编码器实现了对数据的特征抽象和数据降维,因此需保证输入的数据具有一定的 特征 ,即数据之间不是独立同分布的.如果每一个输入都是特征完全无关的,那么就会导致很难学习到特征.

大脑 计算机视觉 医学影像 深度学习（Deep Learning）

关注者 659 被浏览 592,131

**医学图像分割研究生，想放弃了咋办~?**

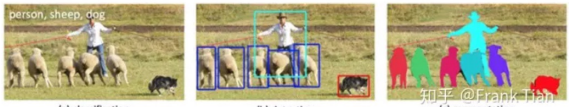
整个实验室就我一个人在做这个方向（其他同学做的也是深度学习的各个方向）；
导师还没有数据集，只能自己在网上找开源数据集;
导师什么建设性的意见都没有;
我自己能力也到顶了;
自己配的一台小小型的电脑;
怎么办，我的两篇北大核心，何时才能发啊~
（不要骂，我真的努力了）
（太难了啊）

关注问题 写回答 邀请回答 好问题 96 9 条评论 分享 ··· 收起

**不是天真是无邪**

开源数据集很多的 MICCAI历年的challenge自己看看 总有一款适合你 多跑实验就完事了 一学期水一篇ei不过分的吧

2021-04-10 问复 喜欢




(a) classification (b) detection (c) segmentation

上面的图片中，分别是计算机视觉的三类任务：分类，目标检测，实例分割。

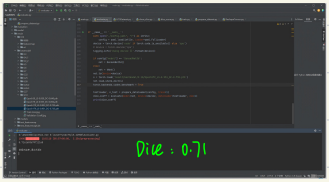
- 白盒攻击，称为White-box attack，也称为open-box 对模型和训练集完全了解，这种情况比较简单，但是和实际情况不符合。
- 黑盒攻击，称为Black-box attack，对模型不了解，对训练集不了解或了解很少。这种攻击和实际情况比较符合，主要也是主要研究方向。
- 定向攻击，称为targeted attack，对于一个多分类网络，把输入分类误判到一个指定的类上
- 非定向攻击，称为non-target attack，只需要生成对抗样本来欺骗神经网络，可以看作是上面的一种特例。

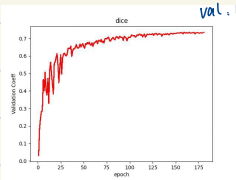
# 结果记录

## 5.22 Mon. Vnet

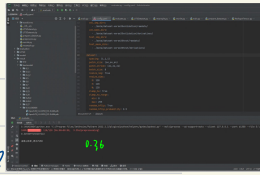
(64, 64, 64)
batch size = 8
170 epoch

9月10日



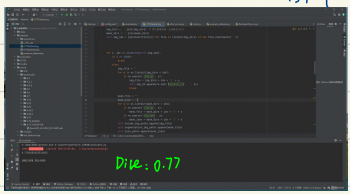

val : 0.73

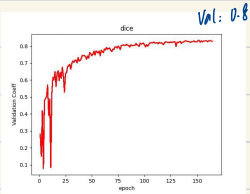



在128 128 上跑

在256 256 256. 跑了10张, 差不多0.3~0.6左右

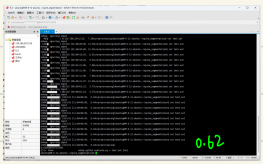
(128, 128, 128)
batch-size = 2
159 epoch




Val: 0.83

在256, 256, 256上跑:



to 是 nn.Moudle的一个方法，要想使用需要从 nn.Mouldle继承

转为tensor放在cuda上跑实例（切patch为例，但最后cpu也爆了，跑的还不如CPU）。

```python
itk_img = torch.tensor(itk_img).float()
# itk_img = itk_img.unsqueeze(0)
itk_mask = torch.tensor(itk_mask).float()
# itk_mask = itk_mask.unsqueeze(0)
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
itk_img = itk_img.to(device)
itk_mask = itk_mask.to(device)
from dataset.GetPatch import GetPatch
method = GetPatch(self.patch_size, self.patch_stride).to(device)
self.preprocessed.append(method(ID, itk_img, itk_mask))
```

将参数转为tensor

设置GetPatch的实例，

并转到GPU上

每次循环都会调用该类的forward方法

```python
from torch import tensor
from torch import nn


class GetPatch(nn.Module):
    def __init__(self, patch_size: list, patch_stride: list):
        super(GetPatch, self).__init__()
        self.patch_size = patch_size
        self.patch_stride = patch_stride

    def forward(self, ID,  itk_img: tensor, itk_mask: tensor):
        for i in range(0, itk_img.shape[0] - self.patch_size[0], self.patch_stride[0]):
            for j in range(0, itk_img.shape[1] - self.patch_size[1], self.patch_stride[1]):
                for k in range(0, itk_img.shape[2] - self.patch_size[2], self.patch_stride[2]):
                    img_patch = itk_img[i:i + self.patch_size[0], j:j + self.patch_size[1], k:k + self.patch_size[2]]
                    mask_patch = itk_mask[i:i + self.patch_size[0], j:j + self.patch_size[1], k:k + self.patch_size[2]]
                    # show_slices(ID, i, j, k, img_patch)
                    direction = [i, j, k]
                    yield ID, direction, img_patch, mask_patch
```

继承nn.Module类，继承到to的方法

初始化类

相当于GetItem

这个相当于return，只是每次运行到这里都会有返回值，一个函数可以多次返回

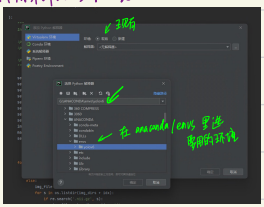
```
nohup: ignoring input
 24%|▌          | 34/141 [01:14<03:54,  2.19s/preprocessing]
Traceback (most recent call last):
  File "main.py", line 12, in <module>
    train_loader = DataLoader(dataset=PatchDataset(config['pathing'][train_img_dirs'],
  File "/home/ubuntu/spine_segmentation/dataset/preprocess.py", line 84, in __init__
    itk_mask = itk_mask.to(device)
RuntimeError: CUDA out of memory. Tried to allocate 176.00 MiB (GPU 0; 14.76 GiB total capacity; 13.49 GiB already allocated; 175.75 MiB free; 13.55 GiB reserved in total by PyTorch)
```

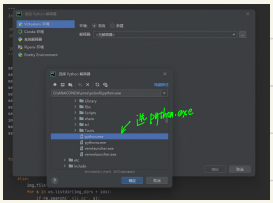
还是爆了。

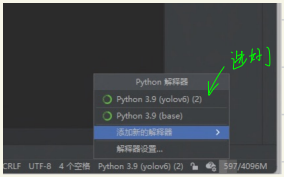
rpm -qa | grep vim 查看vim是否安装

# 新建/选择环境

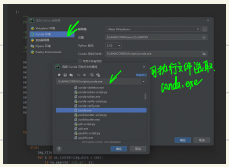
选择原来的环境

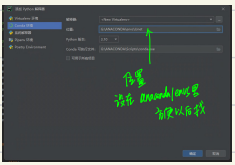


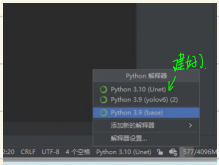




新建环境



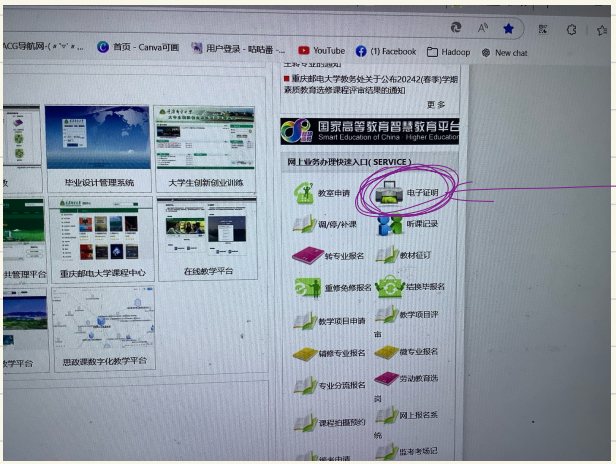

查看安装的库: conda list

激活环境 conda activate 环境名

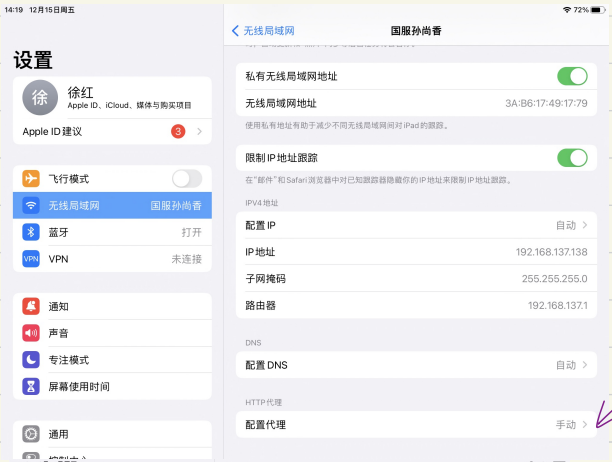
环境 conda env

运用 requirement 安装项目要求的库

pip install -r requirement.txt




点击单条记录可查看详细




手动

复制过来

clash 上看的

cd.. 返回上一级目录

在Hadoop中，`fs` 是一个Hadoop文件系统（HDFS）的Shell命令。它用于执行与文件系统相关的操作，例如浏览目录、复制文件、删除文件等。

`hadoop fs` 命令的基本语法是：

```bash
hadoop fs [generic_options] -command [command_options]
```

其中，`[generic_options]` 是一些通用的Hadoop命令选项，而 `-command` 指定要执行的具体文件系统命令，`[command_options]` 是该命令的选项和参数。

根路径

必须要带路径，不然就写成 hadoop fs -ls /

例如，常见的 `fs` 命令包括：

- `hadoop fs -ls <path>`：列出指定路径下的文件和目录。
- `hadoop fs -cat <path>`：显示文件的内容。
- `hadoop fs -copyToLocal <src> <dest>`：将HDFS中的文件复制到本地文件系统。
- `hadoop fs -copyFromLocal <src> <dest>`：将本地文件复制到HDFS。
- `hadoop fs -mkdir <dir>`：在HDFS中创建一个新目录。
- `hadoop fs -rm <path>`：删除指定路径的文件或目录。
- 等等。

`fs` 命令允许用户在Hadoop集群上执行文件系统操作，通过这些命令，用户可以管理Hadoop分布式文件系统中的数据。

同时支持

hdfs dfs { -put <本地> <远端> 类似 copyFromLocal
-get <远端> <本地>

You

drwxr-xr-x - root supergroup 0 2024-03-12 13:51 /output/1

ChatGPT

这是一个在Hadoop分布式文件系统（HDFS）中显示的目录信息。让我们解释这个输出：

- `drwxr-xr-x`：这表示是一个目录。每一位字符代表不同的权限。在这里，`d` 表示目录。
- `root`：这是该目录的所有者。
- `supergroup`：这是该目录的所属组。
- `0`：这是该目录的大小，对于目录，通常显示为0。
- `2024-03-12 13:51`：这是目录的最后修改时间。
- `/output/1`：这是目录的路径。

综合起来，这行的含义是，`/output/1` 是一个目录，由 `root` 用户拥有，属于 `supergroup` 组，大小为0，最后修改时间为 2024-03-12 13:51。目录权限为 `drwxr-xr-x`，表示 `root` 用户有读、写、执行权限，`supergroup` 组有读和执行权限，其他用户也有读和执行权限。

如果有其他文件或子目录，它们将在这个目录下列出，并且每一行的开头都会显示类似的信息，以描述各个文件或子目录的属性。

```
Found 8 items
drwxr-xr-x   - root supergroup          0 2024-03-12 13:38 /GSOD
drwxr-xr-x   - root supergroup          0 2024-03-12 13:51 /GSOD_ALL
drwxr-xr-x   - root supergroup          0 2021-05-09 23:10 /hbase
drwxr-xr-x   - root supergroup          0 2024-03-12 14:05 /ncdc
-rw-r--r--   3 root supergroup        250 2024-03-12 13:42 /ncdc_files.txt
drwxr-xr-x   - root supergroup          0 2024-03-12 14:43 /output
drwx------   - root supergroup          0 2021-03-25 15:10 /tmp
drwxr-xr-x   - root supergroup          0 2021-03-25 15:10 /user
[root@master data]#
```

生成 MapReduce 的 input 文件

- generate_input_list.sh文件的内容如下：

```bash
#! /bin/bash
a=$1
rm  ncdc_files.txt
./hdfs  dfs  -rm  /ncdc_files.txt

while [ $a  -le  $2 ]
do
        filename="/GSOD/${a}/gsod_${a}.tar"
        echo -e "$filename" >>ncdc_files.txt
        a=`expr  $a  +  1`
done

./hdfs  dfs  -put  ncdc_files.txt  /
```

→ 指定脚本使用的解释器

ChatGPT

这个 Bash 脚本的作用是生成一个包含一系列文件路径的 `ncdc_files.txt` 文件，并将它上传到 HDFS 中的根目录（`/`）。

让我为你解释一下脚本的每个部分：

1. `a=$1`：将脚本的第一个参数赋值给变量 `a`，这里假设你会传递两个参数给脚本，分别是起始值和结束值。
2. `rm ncdc_files.txt`：删除当前目录下的 `ncdc_files.txt` 文件。
3. `./hdfs dfs -rm /ncdc_files.txt`：删除 HDFS 中的 `/ncdc_files.txt` 文件。
4. `while [ $a -le $2 ]`：一个循环，从 `$1`（起始值）一直循环到 `$2`（结束值）。
5. `filename="/GSOD/${a}/gsod_${a}.tar"`：构建文件路径，其中 `${a}` 被替换为循环变量的当前值。
6. `echo -e "$filename" >>ncdc_files.txt`：将文件路径追加到 `ncdc_files.txt` 文件中。
7. `a=`expr $a + 1``：将循环变量 `a` 增加 1。
8. `done`：结束循环。
9. `./hdfs dfs -put ncdc_files.txt /`：将本地的 `ncdc_files.txt` 文件上传到 HDFS 的根目录。

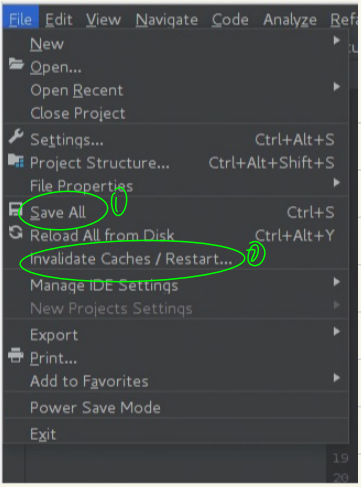
请确保以下几点：

- Hadoop 的 `hdfs` 命令和脚本所在的路径是正确的。
- 你有适当的权限执行这个脚本。
- 脚本的参数 `$1` 和 `$2` 已经正确传递。

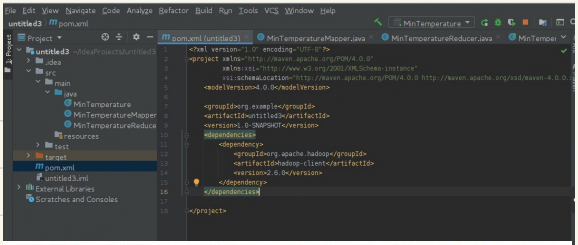
运行脚本时，你可以通过类似以下方式传递参数：

```bash
bash your_script.sh start_value end_value
```

请将 `your_script.sh` 替换为你脚本的实际文件名，`start_value` 和 `end_value` 替换为你希望的起始和结束值。




改完Xml后先 Save all
再 restart
使其依赖加载。




这是一个 Maven 项目的 `pom.xml` 文件，用于管理项目的依赖项。让我为你解释其中的一些关键元素：

1. `<groupId>`：定义项目的组织或者包名。在这个例子中，`org.example` 是组织或包的标识符。
2. `<artifactId>`：定义项目的实际项目名或者模块名。在这个例子中，`untitled3` 是项目的标识符。
3. `<version>`：定义项目的版本号。在这个例子中，版本号是 `1.0-SNAPSHOT`。
4. `<dependencies>`：定义项目的依赖项。在这个例子中，有一个依赖项：

```xml
<dependency>
    <groupId>org.apache.hadoop</groupId>
    <artifactId>hadoop-client</artifactId>
    <version>2.6.0</version>
</dependency>
```

这表示项目依赖于 Apache Hadoop 的客户端库，版本为 `2.6.0`。

这个 `pom.xml` 文件描述了 Maven 项目的基本信息和依赖关系，使得 Maven 能够管理和构建项目。在构建项目时，Maven 将下载并添加所需的依赖项，确保项目可以顺利地构建和运行。

echo → print

$ + 变量 → 变量的值

如：

| |
|---|
| a=1 |
| echo $a |

输出 1

| |
|---|
| echo a |

输出 a.

等号两边不能有空格！

查看 Hadoop 配置文件 core-site.xml (可以看到端口号

echo $HADOOP_HOME

↓进去后

ect

↓

hadoop

$f(0)=0.$

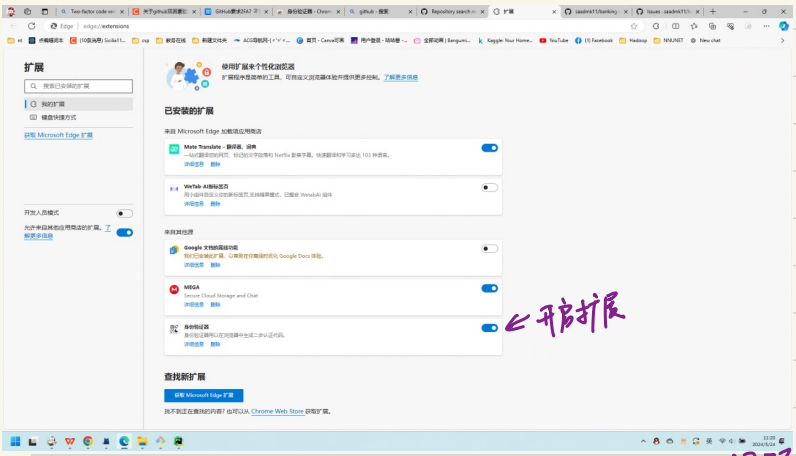
$$\frac{f(x)-f(0)}{x-0}=\frac{f(x)}{x}=f'(x)$$

$$\frac{f'(x)}{3x^2}$$

# github 2-FA 双重身份验证

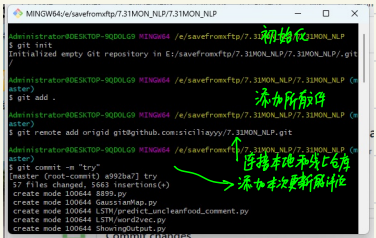
身份验证器网址

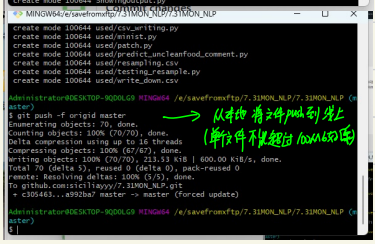
https://chrome.google.com/webstore/detail/authenticator/bhghoamapcdpbohphigoooaddinpkbai/related



# 将文件上传至Github上

强行push方法




在C++中，头文件、源文件和外部文件有不同的作用和特点：

1. **头文件（.h或.hpp）：**
   - **作用**：声明函数、类、变量和宏等。头文件通常包含接口信息，供源文件引用。
   - **特点**：通过`#include`指令包含在源文件中，以便源文件能够使用头文件中声明的内容。
2. **源文件（.cpp或.cxx）：**
   - **作用**：定义函数、类成员函数等具体实现。源文件包含了头文件中声明的具体实现代码。
   - **特点**：通常包含`#include`指令来引用相关的头文件，并实现头文件中声明的功能。

为什么在visual studio里把项目工程叫做解决方案

在Visual Studio中，"解决方案"（Solution）是一个重要的概念，用于组织和管理多个项目。下面是一些关键点，解释为什么使用"解决方案"这一术语：

1. **组织结构**：一个"解决方案"可以包含多个项目（Project），每个项目可以是一个独立的应用程序、库或组件。通过将相关项目组合到一个解决方案中，你可以更好地管理和组织它们。解决方案使得这些项目可以一起开发、构建和调试。
2. **项目依赖关系**：在一个解决方案中，你可以设置项目之间的依赖关系。例如，一个项目可能依赖于另一个项目中的某些功能或库。解决方案帮助你处理这些依赖关系，并确保在构建时按照正确的顺序进行构建。
3. **构建配置**：解决方案管理不同的构建配置，如调试（Debug）和发布（Release）。你可以为整个解决方案配置不同的构建选项，确保各个项目在不同的配置下都能正确构建和运行。
4. **开发效率**：解决方案提供了一种集中的方式来管理和维护多个项目的设置和选项。这种组织方式有助于提高开发效率，使得开发人员能够更轻松地进行版本控制、调试和维护。
5. **集成工具**：Visual Studio的许多工具和功能，如代码分析、单元测试和项目管理，都是围绕解决方案进行组织的。解决方案作为一个整体，使得这些工具能够更好地与各个项目进行集成。

总结来说，"解决方案"这一术语在Visual Studio中用于描述一个容器，它包含了一个或多个项目，并提供了一个统一的管理和组织框架，以支持复杂的软件开发工作。

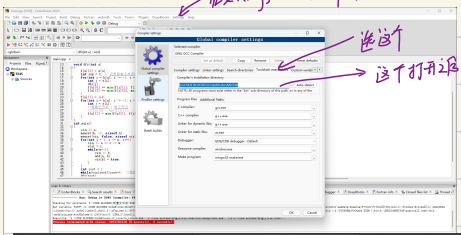
- .sln 就是 Visual, studio 的解决方案文件

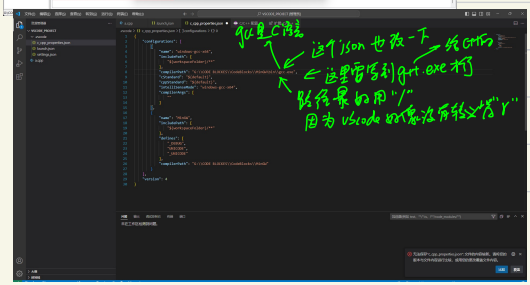
# vscode 配置编译器

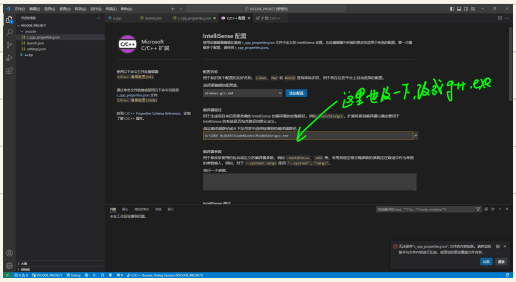
gcc 编译器

妈的! vscode 的 C++ 配置没有自带的编译器，于是我只想着把 code block 的编译器拿给 vscode 使用

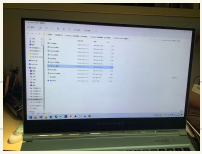
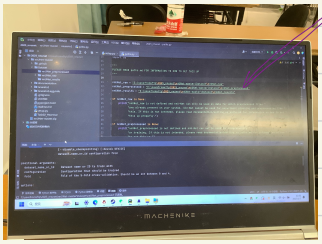
C++ 直接用 g++.exe 编译器



py charm 破解.



nnunet, nnunet-train 显示读不到 nnunet_raw
nnunet_preprocessed 路径
在这里改