# 编程基础

## 基础排序

### 计数排序

计数排序的思路：首先计算一个数组的元素名次，然后使用一个临时数组按照名次排名依次放入对应的原数组元素，再把这个临时数组按顺序赋值给原数组进行覆盖即可，比较的次数n(n-1)/2，原数组->临时数组->原数组赋值次数共计2n，总的时间复杂度为2n+n(n-1)/2。

名次的计算规则是：一个序列中某个元素的名次等于所有比它小的元素个数总数+它左边和它相同的元素个数，也就是不仅要考虑比别人大，也要考虑与相同大小元素相比出现的相对名次。例如数组[4,3,9,3,7]，比4小的只有2个3且左边没有4所以排名第2，7排名3，9排名4，而2个3大小相等就考虑谁先出现，第1个3先出现所以名次靠后为0，第2个3名次为1，最终返回值为[2,0,4,1,3]。

#### C++实现

##### 常规实现

常规实现需要临时数组，如下所示。

template<class T>

void count\_sort(T a[], int n, bool reverse, bool show)

{

int count = 0;

T\* rank = new T[n]; // rank用于得到名次

for (int i = 0; i < n; i++)

rank[i] = 0; // 名次先全部初始化为0

for (int i = 1; i < n; i++)

for (int j = 0; j < i; j++)

{

count++;

if (a[j] <= a[i]) rank[i]++;

else rank[j]++;

}

assert(count == n \* (n - 1) / 2);

T\* temp = new T[n]; // temp用于存放排序后的数组

for (int i = 0; i < n; i++)

{

temp[rank[i]] = a[i]; // rank[i]是a[i]被移动的新位置

}

if (!reverse) // reverse=false表示升序 从小到大

{

if (show)

{

for (int i = 0; i < n; i++)

std::cout << "before => a[" << i << "] = " << a[i] << " new index = " << rank[i] << " after => temp[" << rank[i] << "] = a["<<i<<"] = " << temp[rank[i]] << std::endl;

}

for (int i = 0; i < n; i++)

a[i] = temp[i]; // 把这样的顺序放回原来的a

}

else

{ // 降序

if (show)

{

for (int i = 0; i < n; i++)

std::cout << "before => a[" << i << "] = " << a[i] << " new index = " << rank[i] << " after => temp[" << rank[i] << "] = a[" << i << "] = " << temp[rank[i]] << std::endl;

}

for (int i = 0; i < n; i++)

a[i] = temp[n - 1 - i];

}

if (show)

{

for (int i = 0; i < n; i++)

std::cout << "a[" << i << "] = " << a[i] << " ";

std::cout << std::endl;

}

delete[] temp;

delete[] rank;

}

##### 优化实现

优化实现无需临时数组，原地重排。

原地重排可以不借助辅助数组temp，同样需要先计算名次rank。然后得到的新名次是无序的，如果名次变得有序，并对数组实施对应的交换，即可让数组有序。交换的位置要注意是i和rank[i]，其中i是当前a[i]的实际位置,rank[i]是a[i]移动去往的位置。所以交换rank本身的2个名次也是交换rank[i]和rank[rank[i]]。代码如下所示。

template<class T>

void situ\_rearrangement(T a[], int n)

{

int count = 0;

T\* rank = new T[n]; // rank用于得到名次

for (int i = 0; i < n; i++)

rank[i] = 0; // 名次先全部初始化为0

for (int i = 1; i < n; i++)

for (int j = 0; j < i; j++)

{

count++;

if (a[j] <= a[i]) rank[i]++;

else rank[j]++; // 得到名次

}

for (int i = 0; i < n; i++)

{

while (rank[i] != i) // 如果名次和实际索引顺序相同则i++检测下一轮

{

int t = rank[i]; // 如果名次和实际顺序不等,记录名次指定的位置

swap\_array\_element1(a, i, t); // 首先把a的当前元素交换到名次指定的位置

swap\_array\_element1(rank, i, t);//a发生了变化,名次也发生了变化,让rank[i]和rank[rank[i]]交换即可

// rank记录的是名次位置,当前a[i]移动到a[rank[i]]=a[t]

// 当前rank[i]的值是t=rank[i](原有名次)应当和实际索引i交换

}

}

for (int i =0 ; i < n ; i++) // rank的顺序由计算的名次恢复为顺序名次

std::cout << "rank[" << i << "] = " << rank[i] << " ";

std::cout << std::endl;

delete [] rank;

}

#### Python实现

##### 常规实现

##### 优化实现

### 选择排序

时间复杂度是n(n-1)/2+3(n-1)，比计数排序还要多n-1

选择排序是首先整个数组查找最大元素的位置，然后把这个位置的元素移动到末尾；然后在余下的n-1个元素中继续寻找最大位置，然后移动到倒数第2个位置；如此反复，可以得到升序结果，如果需要降序，就放在开头、第2个、第3个...也是可以的。

#### C++实现

##### 常规实现

常规实现在数组有序时查找最大值索引仍会继续执行。

template<class T>

void select\_sort(T a[], int n, bool reverse , bool show )

{

// 升序：依次在区间[0,n-1],[0,n-2],...[0,1]寻找最大值索引,然后把最大值放在末尾

// 降序：依次在区间[0,n-1],[1,n-1],[2,n-1],...[n-2,n-1]寻找最大值索引, 然后把最大值放在开头j的位置

// 升序时j的区间是[1,2],[1,3],[1,4],...,[1,i](i=n-1结束) 令maxLoc=0,从1开始是为了少比一次 , j ≤n-1所以j<=i or j < i+1

// 降序时j的区间是[i-1,n-2],...,[n-4,n-2],[n-3.n-2](i=1开始) 令maxLoc=n-1,到n-2结束是为了少比一次, j<=n-2 or j <n-1

// 降序写在一个循环里,i从n-1,n-2,...,1变化j从0,2..n-2变化,关系满足i+j=n-1,所以j=n-1-i开始 开始的位置就是最大值存放的位置

for (int i = n-1; i > 0; i--) // i从位置n-1开始 , 可以取到1

{

int maxLoc ;

T temp;

if (!reverse)

{

maxLoc = 0;// 先将第一个元素认为是最大值的索引

// i取n-1时j要取等号防止j取不到，j要从1开始可以少一次a[0]和a[0]比较

for (int j = 1; j <i+1; j++) // 因为 j <=n-1 , 故j <=i or j<i+1

{

if (a[j] > a[maxLoc]) // j <= n- 1保证不越界

maxLoc = j;

}

temp = a[i];

a[i] = a[maxLoc];

a[maxLoc] = temp;

if (show)

{

for (int i = 0; i < n; i++)

std::cout << "a[" << i << "] = " << a[i] << " ";

std::cout << std::endl;

}

}

else

{

// [0, n - 1] , [1, n - 1], [2, n - 1], ...[n - 2, n - 1] j = 0,1,2,...n-2

maxLoc = n - 1;

// 要求j从n-1-i开始直到n-2，每个小数组都认为n-1是最大的然后进行比较

for (int j = n-1-i ; j < n-1; j++) // j<=n-2 or j <n-1 i=1时j=n-2

{

if (a[j] > a[maxLoc]) //

maxLoc = j; // j可取到1~n-1

}

temp = a[n-1-i]; // 每次i循环开头位置应当是j的位置也就是n-1-i

a[n - 1 - i] = a[maxLoc];

a[maxLoc] = temp;

if (show)

{

for (int i = 0; i < n; i++)

std::cout << "a[" << i << "] = " << a[i] << " ";

std::cout << std::endl;

}

}

}

}

##### 优化实现

优化实现可以在数组有序时及时终止，当没有发现任何前一个元素大于后于一个元素时就认为有序，否则只要出现就无序。

template<class T>

void select\_sort\_timely\_termination(T a[], int n)

{

bool sorted = false; // 默认无序,无序才会进入循环

for (int k = n; (k > 1) && !sorted; k--) // k取n,n-1,..2=>i取n-1,n-2,..,1

{

int idx = 0; // 比较总是从第1个元素a[1]开始,a[0]默认初始最大值

sorted = true; // 进入循环立刻指定为有序,循环中去判断是否存在前一个元素大于后一个元素

for (int i = 1; i < k; i++)

if (a[idx] <= a[i]) idx = i; // 如果前者小于后者说明还是有序的,并记录最大值位置

else sorted = false; // 任何大循环中即[1,k-1]中存在前者大于后者就说明无序,如果某次大循环已经发现有序则不再进行后续大循环提前终止

T temp = a[idx];

a[idx] = a[k - 1]; // 总是把最大值放在每次大循环的最后

a[k - 1] = temp;

}

}

#### Python实现

##### 常规实现

##### 优化实现

### 冒泡排序

#### C++实现

思想是先比较数组n个元素，从左到右相邻2个元素比较，如果前者大于后者则交换，最终最大的元素在右侧；然后在对数组的前n-1个元素继续进行冒泡，如此循环，直到最后一次冒泡，全部的元素都升序排列。

bubble函数的时间复杂度是n-1(比较)+(交换)，故总的时间复杂度为[(n-1)+(n-2)+...1]+(交换)=n(n-1)/2 +(交换)。

关于交换，bubble函数是无法确定的，因为取决于两两交换的值，如果每次大循环前者一直比后者小，就不用执行交换，时间复杂度=0；如果每次大循环一直都比后者大，每次都要交换，从a[i]和a[i+1]交换持续到a[n-2]和a[n-1]交换，累积为(n-1)+(n-2)+..+1=n(n-1)/2次，又因为交换函数需要三次赋值过程，故时间复杂度等于n(n-1)/2 \* 3。

##### 常规实现

和选择排序类似，常规实现是数组有序时仍然会两两比较判断一次，虽然不执行交换过程，但是比较判断的这部分时间复杂度可以优化中实现。

template<class T>

void bubble\_sort(T a[], int n, bool reverse, bool show)

{

for (int i = n; i > 1; i--) // i = n,n-1,...2(共n-1次)

{

if (!reverse)

{

for (int j = 0; j < i - 1; j++) // j = 0,1,..i-2=n-2 (共n-1次) a[j+1]最大为a[n-1]没有越界

{

//printf("i = %d j = %d\n", i ,j);

if (a[j] > a[j + 1]) // 前者比后者大把前者(大的)换到后面=>升序

{

T temp = a[j];

a[j] = a[j + 1];

a[j + 1] = temp;

}

}

if (show)

{

for (int i = 0; i < n; i++) // 展示一次冒泡的过程

std::cout << "a[" << i << "] = " << a[i] << " ";

std::cout << std::endl;

}

}

else

{ // 比较相邻元素时可以把小的换到后面实现降序不必像选择排序那样麻烦

for (int j = 1; j < i;j++) // j取1,2,3,..n-1,j-1取0,1,2..,n-2

if (a[j - 1] < a[j]) // 前者比后者小把后者(大的)换到前边=>降序

{

T temp = a[j-1];

a[j-1] = a[j];

a[j] = temp;

}

if (show)

{

for (int i = 0; i < n; i++) // 展示一次冒泡的过程

std::cout << "a[" << i << "] = " << a[i] << " ";

std::cout << std::endl;

}

}

}

}

##### 优化实现

本函数的改进在于如果已经有序，就不必进入下一个大循环继续比较大小，当然更无需交换，所以节省的时间复杂度要注意只有比较的占用时间，因为有序状态时都不会执行交换。

template<class T>

void bubble\_sort\_timely\_termination(T a[], int n)

{

bool swapped =true; // 默认无序交换过

for (int k = n; (k > 1) && swapped; k--) // 没有发生交换就提前终止,否则执行a[0]都a[k-1]之间的元素两两交换

{

swapped = false; // 进入循环立刻先视为有序 后面只有存在前者大于后者就置为true

for (int i = 0; i < n - 1; i++)

if (a[i] > a[i + 1]) // 如果前者元素大于后者就说明无序,此时交换发生swapped=true

{

T temp = a[i];

a[i] = a[i + 1];

a[i + 1] = temp;

swapped = true;

}

}

}

#### Python实现

##### 常规实现

def bubble\_sort(X):

"""

测试程序

x\_sort= np.random.rand(10)

bubble\_sort(x\_sort) # 测试bubble\_sort

y\_sort = (3 \* x\_sort + 4+ np.random.randn(10)\*0.05).reshape(-1,1)

print(y\_sort)

"""

assert X.ndim == 1

for i in reversed(range(X.shape[0]+1)[2:]):# 在区间[0,n-1],[0,n-2],..,[0,1]两两比较

# print(i) # 冒泡排序 i = 100,99,98,...,2

for j in range(i-1): # j = 0,1,2,..i-2 = 0,1,2,...,n-2

if (X[j] > X[j+1]): # 例如当前区间是[0,n-1],则从X[0]和X[1]两两比较直到X[n-2]和X[n-1]

X[j],X[j+1] = X[j+1],X[j] # 如果前者大于后者就交换,比较到最后最大值放在i-1的位置也就是j+1,最大值在X[i-1]上

# 否则进入下一轮小区间比较 依次得到[0,n-2]的最大值在X[n-2],[0,n-3]的最大值在X[n-3]..

# 循环结束 得到升序结果

# 验证是否升序排列

for i in range(X.shape[0])[1:]:

assert X[i-1] <= X[i]

return X

##### 优化实现

### 插入排序

插入排序的思路是从数组的第一个元素开始(第0个元素无需插入)，把a[i]作为插入元素x，然后执行有序数组插入操作，i=1,2,..n-1，所以有序数组插入元素要执行n-1次，i的大小又影响了有序数组插入元素的比较和交换时间复杂度。最终的时间复杂度还是[n(n-1)/2+n-1]/n

#### C++实现

template<class T>

void insert\_sort(T a[], int n, bool reverse, bool show)

{

for (int i = 1; i < n; i++) // a[i]作为x, 从i=1直到n-1

{

T x = a[i]; // 后1个元素

int j = 0;

for (j = i - 1; j >= 0 && x < a[j]; j--) // i-1开始是前一个元素

a[j+1] = a[j]; // i=1=>如果a[1]<a[0],把a[0]换到a[1]的位置上即a[1]=a[0]

a[j + 1] = x;// i=1=>a[0]的位置要放原来的a[1]位置,因为a[0]>a[1]要交换

//推广=>a[i]<a[j]=a[i-1],a[i]的位置应当放置a[i-1]即a[i]=a[i-1]<=>a[j+1]<a[j],然后再把原来a[i-1]放a[i]，即a[j+1]=x

// 如果a[i]>a[j]也就无须执行交换了

// 不可以用i去控制,i控制a[i],a[i]=a[j]会导致外循环出现错误

}

if (reverse)

{

for (int i = 0; i < n / 2; i++)

{ //奇数也没问题

T t = a[i];

a[i] = a[n - 1 - i];//n=10, a[4] = a[10-1-4]=a[5](偶数) 不会取到a[5]=a[4]

a[n - 1 - i] = t; // n=5 a[0]=a[4],a[1]=a[2],会取到a[2]=a[2]不影响结果

}

}

if (show)

for (int i = 0; i < n; i++)

printf("a[%d] = %d ", i, a[i]);

}

#### Python实现

##### 常规实现

##### 优化实现

# Python基础

字符串、元组的元素是不能更改的，列表的元素可以更改

A = (1,2) ,A[0]=1 是错误的

## 列表list

列表可以使用append方法，也可以使用[1]+[2]，也会追加，但是不可以使用[1]+2，触发类型错误

列表也可以添加元组，但是必须用变量，直接[1]+(1,2)是错误的

a = [1]

a += (1,2)

### 排序sort

list.sort(reverse=False) # 默认升序

### 索引index

list.index(x,start,end)

搜索value对应的第一个索引，可以指定开始搜索位置start和结束位置end

### 追加元素append

### 插入insert

list.insert(index, obj)

### 追加序列extend

list.extend(seq)

### 移除元素remove

list.remove(obj)

移除某个元素的第一个匹配项

### 删除元素pop

list.pop([index=-1])

默认移除最后1个元素

### 复制元素\*

a = [1,2,3]

a \* 4

'-'\*20 # 字符串也可以

### 翻转reverse

list.reverse()

### 拷贝copy

list.copy()

### 计数count

list.count(obj)

### 清空clear

list.clear()

## 元组tuple

解包操作，a,b,c=tuple(a,b,c),=[1,2,3]

## 字典dict

无序集合，通过dict[key]来获得键对应的值或者更新，添加键值对直接添加即可，删除键值对则需要借助del删除，生成列表借助list，但是内部元素会变成元组类型

字典推导式：交换key和value

dict2 = {value:key for key,value in dict1.items()}

a = dict(a=2,c=3,e=1)

b = {"a":2,"c":3,"e":1}

### 键keys

dict.values()

### 值values

dict.keys()

### 项items

dict.items()

### 返回键值get

get(key, default=None)

default -- 如果指定键的值不存在时，返回该默认值

### 更新update

可以使用dict.update({‘a’:1,’b’:1})，也可以使用

dict.update(a=1) # 自动将参数名a转为字符键

前提是字典已经存在这个键，如果不存在就要考虑使用defaultdict

### 删除pop

dict.pop(key)

### 清空clear

dict.clear()

### 集合set

## 常见函数

### 打开文件open

with open(file,mode='r',buffering=None,encoding=None,errors=None,newline=None, closefd=True) as f :

f.readlines() # 读取全部文件

f.readline() # 逐行读取

f.read() # 全部读取

# 法1

with open("pycharm快捷键.txt",mode='r',encoding='utf-8') as f :

read = f.read()

for line in read:

print(line)

f.close()

# 法2

for line in open("pycharm快捷键.txt",encoding='utf-8'):

print(line)

f.close()

# 法3

f = open("pycharm快捷键.txt",mode='r',encoding='utf-8')

line = f.readline()

while line :

print(line)

line = f.readline()

# 法4

f = open("pycharm快捷键.txt",encoding='utf-8')

lines = f.readlines()

for line in lines:

print(line)

### 翻转reversed

a=[1,2,3]

b = list(reversed(a))

### 匿名函数lambda

def func(param):

return expression

等价于使用

lambda param: expression

### 过滤filter

filter(function, iterable)

filter(None, square)，打印数值的话需要使用list

lambda x : x != None也是可以的

### 映射map

map(function, iterable, ...)

map(lambda x: x \*\* 2, [1, 2, 3, 4, 5])

### 删除del

x = [0,1]

del x[0] 可以删除元素

del x 也可以删除变量

### 排序sorted

默认升序，若需要降序传递reverse=False即可

### 数组逻辑any/all

any(iterables)和all(iterables)对于检查两个对象相等时非常实用

any()实现了或(OR)运算，而all()实现了与(AND)运算

any：任意存在每一个元素为True则返回True

all：可迭代对象iterables中所有元素都为True则返回True

### 输出print

格式化输出：

epoch = 10

loss = 0.55324

acu = 0.935657

print(f"{epoch} {acu} {loss}")

print(f"{loss:.20f}") # 可以不借助format控制位数

print("{:d} {:.3f} {:.3f}{:s}".format(epoch,acu,loss))

print("%d %.2f %.2f%s" % (epoch,acu,loss))

%s表示字符串，%d表示整数，%f表示小数， %%表示一个百分号

### 范围range

for i in range(10,-2,-2):

print(i) # 倒序

### 继续continue

一般用于在遍历到某个位置忽略，然后继续，break会直接中断循环，continue可以继续后续的循环

for i in range(10):

if i == 5 :

continue

print(i)

### 声明全局global

global x

全局变量可以在函数中作用，如果函数内部创建了同名的变量，会屏蔽掉全局函数，但是函数外部仍然是全局变量且不发生变化，如果希望函数内部改变全局变量，必须事先声明

### 对象标识id

x0 = 7

a = id(x0)

def func(x):

print( x is x0) # True

return id(x)

print(func(x0) == a) # True

Python中所有变量都是对象，都是引用传递，即变量x中其实不包含值7，而是对包含7这个值的对象的引用，通过id函数可以获取这个对象的标识号，标识号是唯一的，或者说占据内存的这个对象是唯一的

函数的参数也是引用传递，func的参数x实际上是对x的拷贝，也是引用，避免直接复制x的内存耗费大量时间

### 标识符是否相同is

X is Y

如果两个操作数具有相同的标识则返回True

### 异常捕获try\_except\_finally

try:

# 代码块

except ValueError as e:

# 出现值异常错误如何处理

print(“请核对输入值”,e)

except ZeroDivisionErrot as e:

# 出现零除如何处理

print(“请输入非零除数”,e)

else：

# 没有异常执行的代码块

finally：

# 一定会执行的代码块，通常用于释放代码资源

except可以捕获多个异常，如果不同类型的错误提示的信息相同，可以使用元组，try (type1,type2,...) as name:

as可选，如果不需要引用捕获的异常对象，不需要as语句

还可配合with语句使用：

try:

with open('123.txt',''r'') as f:

# 代码块

except FileNotFoundError:

print(''The file is not exist'')

使用raise显示扔出异常：

class CustomException(Exception):

pass

def test():

b = 5

try:

# 自己扔出异常

if b < 10:

raise ValueError("b must be > 10")

aaa

raise CustomException

except CustomException:

print('触发自定义异常...')

except NameError:

print("变量名异常")

test() # "b must be > 10"

## 常见技巧

### 带不定长参数\*args

def avarage(\*args):

return sum(\*args)/len(args)

可以分别带入avarage(1,2,3)，也可以直接对可迭代对象解包带入，即avarage(\*[1,2,3])

### 字典参数打包符号\*\*

传入较多字典参数时可以打包，然后使用\*\*即可

例如定义了pacth\_kw=dict{‘color’:’r’,linewidth:2}

plt.plot(…,\*\*patch\_kw),可以传递多个字典参数

### 解决图例中文显示不全

from matplotlib.font\_manager import FontProperties

myfont = FontProperties(fname=r"C:\Windows\Fonts\simsun.ttc", size=10.5)

axe.legend(prop=myfont,loc="upper right")

### 广播效应

两个数组有一方的长度为1，或者两个数组的后缘维度轴长度相同，则是可以广播的对象，例如(4,3)可以和(3,)兼容，(4,3)也可以和(4,1)兼容

广播效应只应用于四则运算，点积和外积不参与，按照运算法则来

X[X>0]=0，数组大于0的元素全变为0

<1> 向量和向量运算(M,)(M,)

四则运算：对应元素加减乘除即可

点积：对应元素相乘后求和，长度不同的无法点积，具有交换性

外积：看成向量a的每个元素对向量b相乘，然后按行堆叠，不具有交换性，np.outer(a,b)和np.outer(b,a)不同

<2>矩阵和向量运算(M,)(M,N),N可以等于1

四则运算：矩阵和行向量运算，必须矩阵的列数和行向量长度相同；矩阵和列向量运算，必须行数和列向量长度相同；

计算方式：矩阵的每行与行向量对应元素相乘，矩阵的每列与列向量对应元素相乘，也就是广播效应，拆分成向量和向量的运算,形状仍是矩阵的形状

点积：依然拆分成矩阵的每行同行向量的点积，矩阵的每列同列向量的点积

外积：依然拆分成矩阵的每行同行向量的外积，矩阵的每列同列向量的外积

<3>矩阵和矩阵运算

必须同型才能四则运算

### 生成指定长度列表

Label = [ [] for \_ in range(10)]

### 列表推导式

b = [i for i in range(15) if i&1] #创建奇数列表

square = [ x \* x for x in range(10)]

[x for x in global\_loss.index() if x % iters == 0]

### 生成器对象

列表推导式无需全部的值时，应考虑惰性计算，生成迭代器，而非贪婪计算

使用小括号而非中括号，即

square = ( x \* x for x in range(10) )

### 计数器collections.Counter

接受1个可递归的对象，汇总并计算元素，返回字典

from collections import Counter

a = [1,2,3,4,3,3,3,2]

counter = Counter(a)

out：Counter({1: 1, 2: 2, 3: 4, 4: 1})g

### 字典对象collections.defaultdict

d = {} # 普通字典对象

for key in ['1','2','3','2','2','3']:

# d[key] +=1 提示报错 不存在key'1'

d[key] = d.get(key,0) + 1 # 可以使用get避免

或者：

from collections import defaultdict

d = defaultdict(int) # 修正字典对象

for key in ['1','2','3','2','2','3']:

d[key] +=1

defaultdict(int)是指明如果键不存在的时候，默认返回的对象是int。

例如dict1 = defaultdict(int)，dict1[2]="two"，但是输出dict1[1]就会返回int的默认值0。

### 进度条tqdm

一般常用于读取文件，使用的必须是可迭代是对象

from tqdm import tqdm

from tqdm import trange

j = 0

for i in tqdm(range(100000000),colour='green'):

j += 1

for i in trange(10000000):

j += 1

// 也可以使用自定义的信息处理set\_description

j = 0

bar = tqdm(range(100000000),colour='green')

for i in bar:

j += 1

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

// 还可以使用with\_update结构

with tqdm(total=100) as pbar:

for i in range(100):

time.sleep(0.05)

pbar.update(1)

// 使用set\_postfix设置进度条显示信息

with trange(100) as t:

for i in t:

# 设置进度条左边显示的信息

t.set\_description("epoch %i" % i)

# 设置进度条右边显示的信息

t.set\_postfix(loss=random(), acu=randint(1,5), str="h", lst=[1, 2])

time.sleep(0.1)

可以这样打印信息，使用sys.stdout可以实时刷新终端。

   train\_bar = tqdm(train\_data\_gen, file=sys.stdout)

   for images, labels in train\_bar:

        train\_step(images, labels)

        # 属性设置或者使用方法都可以

        #  train\_bar.desc="train\_loss:{:.3f},train\_acc:{:.3f}".format(train\_loss.result(),train\_accuracy.result())

        train\_bar.set\_description("epoch [{}/{}] loss: {:.3f}, acc: {:.3f}".format(epoch+1,epochs,train\_loss.result(),train\_accuracy.result()))

打印效果：

epoch [1/5] loss: 3.015, acc: 0.205: 59%|███████████▏ | 54/92 [05:51<04:38, 7.34s/it]

### 相同规律打乱列表

np.random.shuffle(X)

np.random.set\_state(state) # 保证样本和标签以相同的规律被打乱

np.random.shuffle(y)

### bool数组切片索引

a = np.arange(12).reshape(3,4)

b = a > 4

c = a[b]返回大于4的内容

典型的应用是在训练数据中找到指定标签的所有数据

label = 0

Xlabel = X[y ==label,:0]

### 整型数组切片索引

step = np.arrange(0,21)

step[np.array([2,4,6,8])] # 返回索引位置为2,4,6,8的数

### 保存sklearn模型

#### 使用joblib

import joblib

# joblib.dump(clf,'mnist\_svm.joblib')

model = joblib.load(root+'\\9.经典机器学习方法\\'+'mnist\_svm.joblib')

#### 使用pickle

import pickle

path = root+'\\9.经典机器学习方法\\'+'mnist\_svm.pickle'

with open(path, 'wb') as f:

pickle.dump(clf, f)

with open(path, 'rb') as f:

model\_ = pickle.load(f)

#### 评估运行时间%time/%timeit

%time指一次执行代码的总体时间

%timeit指多次执行代码的平均时间

%hist指历史命令

%xdel variable 删除单个变量的引用

%reset指删除interactive命名空间中全部的变量名

%run命令执行脚本文件

%logstart记录日志

相对应的日志命令还有%logstop，%logon，%logoff，%logstate

# python库

## os库

### path模块

### 分割目录spilit

os.path.split(path)[-1] # 得到最后一级文件名称

返回的是分割后剩下的倒数1级文件名称和父目录，如果想得到倒数第2级要重复操作

### 列出文件夹listdir

os.listdir(flower\_dataset\_path)

### 是否为文件夹isdir

os.path.isdir(os.path.join(flower\_dataset\_path,dir\_name))

### 拼接join

os.path.join(flower\_dataset\_path,dir\_name)

## Operator库

存在3个类， attrgetter , itemgetter ,methocaller

### attrgetter模块

### itemgetter模块

这个类会返回一个可调用对象，它会从它的操作数里面取值,会调用操作数的\_\_getitem\_\_()方法获取值。如果传入了多个 item , 那么结果就会返回一个元祖。

可以用于对字典排序，itemgetter(1)表示从dict的第1个索引处也就是value获取值

sorted(dict.items(),reverse=True,key=operator.itemgetter(1))

a = {"a":2,"c":3,"e":1}

# b = sorted(a.items(),reverse=True,key=a.values()) # 想要按照字典的value排序,单数a.values()是不可迭代的会报错

# 如果给定item就会自动抓取返回可迭代元组对象

# operator.itemgetter()可以理解为可迭代的(key、value)的操作对 常用于对字典排序

b = sorted(a.items(),reverse=True,key=operator.itemgetter(0)) # 0表示按照key排序,返回元组对的列表

c = sorted(a.items(),reverse=True,key=operator.itemgetter(1)) # 1表示按照value排序,返回元组对的列表

### methocaller模块

## Random库

### 范围内随机整数randrange

返回指定范围内的随机整数

randrange(start=1, stop=7, step=2)

### 随机种子seed

保证可重复性

### 高斯随机数guass

random.gauss(0,1)

## Numpy库

### fft模块

与scipy.signal的fft用法相同

### linalg模块

#### 矩阵求逆inv

inv(a)

### random模块

#### 随机抽样choice

choice(a,size,replace=True)

a可以是一维向量，a中抽取size个元素返回数组

a可以是整数，自动生成0到a的整数序列进行抽取

replace表示是否放回抽样，默认是True

#### 均匀分布rand

np.random.rand(\*dn)

#### 均匀分布uniform

np.random.uniform(0,1,10)

#### 高斯分布randn

np.random.randn(\*dn)

#### 高斯分布normal

np.random.normal(0,1,10)

#### 随机整数randint

# 默认生成(0,low)之间的整数，指定high则是[low,high)之间的整数

# 默认size=None，返回1个数，也可以size=10返回10个数,还可以size=(5,3)指定形状的数组

np.random.randint(10, high=None, size=(2,6),dtype=np.int32)

#### 打乱shuffle

np.random.shuffle(x)

#### 获取/设置状态get\_state/set\_state

可以用于保证相同随机规律

state = np.random.get\_state()

A = np.arange(60000)

B = np.arange(60000)

np.random.shuffle(A)

np.random.set\_state(state)

np.random.shuffle(B)

#### 设置种子seed

设置种子也可以用于保证相同随机规律

A = np.arange(60000)

B = np.arange(60000)

np.random.seed(0)

np.random.shuffle(A)

np.random.seed(0)

np.random.shuffle(B)

### 常用函数

#### 创建数组索引nditer

layer\_w = np.random.rand(50,50)

genegrator = np.nditer(layer\_w,flags=['multi\_index'], op\_flags=['readwrite'])

while not genegrator.finished :

print(genegrator.multi\_index)

genegrator.iternext()

flags=['multi\_index']表示对数组进行多重索引

op\_flags=['readwrite']规定迭代器的权限

#### 拆分数组array\_split

array\_split(ary, indices\_or\_sections, axis=0)

将一个数组拆分为多个子数组

x = np.arange(7.0)

np.array\_split(x, 3)

#### 重复tile

默认情况下np.tile(A,2)表示对A在X轴(axis=0 or -1)方向平铺2倍,或者说对A按列堆叠

a = np.array([[1],[2]]) # (2,1)

b = np.tile(a,3)

out: (2,3)的array([[1, 1, 1],

[2, 2, 2]])

b = np.tile(a,(2,3))表示在Y轴方向(axis=1)对a平铺2次，X轴(axis=0 or -1)方向平铺3次

out: array([[1, 1, 1],

[2, 2, 2],

[1, 1, 1],

[2, 2, 2]])

注意：reps元素最后1个数才是轴0(X轴列方向)的平铺倍数,与np.mean等的参数axis=0含义类似。axis=-1表示最后1个维度,也就是Y维度,在numpy中Y维度是行方向(样本方向)

#### 扩充pad

pad(array, pad\_width, mode, constant\_values=int)

pad\_width ：{sequence, array\_like, int}

((before\_1, after\_1), ... (before\_N, after\_N))，对每个轴进行填充

mode：函数或者str，默认constant，即常数填充；也可以使用edge，使用数组的边值填充；linear\_ramp，在end\_值和阵列边缘值之间使用线性渐变进行填充；maximum，沿每个轴的全部或部分向量的最大值的填充，类似的还有mean、median、minimum；其他还有reflect、symmetric、wrap、empty

constant\_values：序列或者标量，((before\_1, after\_1), ... (before\_N, after\_N))对应每个轴，使用((before, after),)则每个轴使用的相同，如果使用(constant,)或者'constant'则所有轴都使用相同的元素，默认0填充

对一维：pad\_width表示左右填充

a = [1, 2, 3, 4, 5]

np.pad(a, (2, 3), 'constant', constant\_values=(4, 6))#左右填充2,3列

out：array([4, 4, 1, ..., 6, 6, 6])

对二维：pad\_width表示上下、左右填充

a = [[1, 2], [3, 4]]

np.pad(a, ((4, 2), (1, 3)))

#### 深拷贝copy

浅拷贝可以使用view函数

b = a.view() # 两个不同的对象但是指向同一块内存，即引用传递

b = a.copy() # 不指向同一内存，只对array对象拷贝

如需对其他的python类型进行拷贝，使用copy模块的deepcopy

import copy

copy.deepcopy()

#### 返回升序索引argsort

argsort(a, axis=-1, kind=None, order=None)

多维数组时指定按axis轴进行排序，kind指定排序算法，可选

{'quicksort', 'mergesort', 'heapsort', 'stable'}

用处：可以得到一个列表的升序/降序索引对另一个列表进行映射排序

argsort函数的作用是首先对序列升序,再遍历每个元素得到原来序列的实际索引位置

例子：

np.argsort([3,1,2,4])

array([1, 2, 0, 3], dtype=int64)

[3,1,2,4]升序后是[1,2,3,4],开始遍历1的实际位置是1,2的实际位置是2,3的实际位置是0,4的实际位置是3。所以输出为[1, 2, 0, 3]

#### 计算相关系数corrcoef

corrcoef(x, y=None, rowvar=True, bias=np.\_NoValue, ddof=np.\_NoValue)

x：包含多个变量和观测值的1D或2D数组，每行代表1个变量，每列代表变量的观察值

y：另外一组变量和观察，y和x有相同的形状

rowvar：默认为True，表示每行代表1个变量，并在列显示，否则转换关系

bias：不使用

ddof：不使用

返回：变量的相关系数矩阵A

把x，y的所有行按顺序堆叠，那么A的第一行表示x的第1行依次和自己，其他行的相关系数，A的第二行表示x的第2行与其他行、自己的相关系数



具体见<https://blog.csdn.net/qq_39514033/article/details/88931639>

#### 绘制二维坐标系meshgrid

返回指定范围内等分的轴点坐标，类似于生成圆周长坐标，可参考生成圆周长坐标circle\_perimeter的使用方式

可以接受两个一维数组生成两个二维矩阵，对应两个数组中所有的(x,y)对

x = np.linspace(0, 255, 256)

y = np.linspace(0, 255, 256)

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

#### 绘制多维坐标系mgrid

返回多维结构，常见的如2D图形，3D图形。对比np.meshgrid，在处理大数据时速度更快，且能处理多维（np.meshgrid只能处理2维）

mgrid在函数内部指定范围且使用中括号，但是用法类似于arange(),按步长给定而非等分为多少份linespace

另外返回的元组是yy\_和xx\_\_，正好相反

x = np.linspace(0, 255, 256)

y = np.linspace(0, 255, 256)

z = np.linspace(0, 255, 256)

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

yy\_,xx\_= np.mgrid[0:255:1,0:255:1]

# ,xx\_和xx一样 yy\_和yy一样

print(bool(bool(yy\_ == yy) == bool(xx\_ == xx))) # True

#### 未初始化空对象empty

np.empty(shape=(4,5,2), dtype=int)

#### 向量外积outer

outer(a,b)

计算两个向量的外积

#### 向量内积dot

dot(a,b)

计算两个向量的内积

#### 上下翻转flupud

整体行向量向上反转

#### 左右翻转fliplr

整体列向量向上反转

#### 翻转flip

axis=0等同于flupud

axis=1等同于fliplr

#### 水平垂直拼接stack

vstack垂直堆叠数组，行维度取决于堆叠的数量

hstack水平和堆叠数组，列维度取决于堆叠的数量

a , b ,c = np.array([1,2,3,3]),np.array([1,2,4,5]),np.array([1,2,4,5])

v = np.vstack((a,b,c)) # (3,4)

h = np.hstack((a,b,c)) # (12,)

#### 水平垂直拼接concatenate

concatenate((x,y),axis=0)

axis=0按行拼接，要求列数相同

axis=1按列拼接，要求行数相同

a , b ,c = np.array([1,2,3,3]),np.array([1,2,4,5]),np.array([1,2,4,5])

v = np.vstack((a,b,c))

h = np.hstack((a,b,c))

e = np.concatenate((v,v),axis=0) # (6,4)

j = np.concatenate((v,v),axis=1) # (3,8)

#### 按行拼接np\_r

要求行数相同

b = np.r\_[np.array([1,2,3]).reshape(1,-1),np.array([4,5,6]).reshape(1,-1)]

#### 按列拼接np\_c

要求列数相同

a = np.c\_[np.array([1,2,3]).reshape(-1,1),np.array([4,5,6]).reshape(-1,1)]

常常用于绘制网格

np.c\_[xx.ravel(), yy.ravel()]

#### 裁剪clip

clip(a=im,a\_max=0,a\_min=255,out=None)

将数组限制在指定范围

#### 误差容许allclose

allclose(im1,im2)

如果两个数组在容差内按元素相等，则返回 True

#### 转为数组asarray

将输入转换为ndarray

#### 条件返回where

np.where(condition, [x, y]) 根据“条件”返回从“x”或“y”中选择的元素

eg：

return 20 \* np.log10(np.where(input\_spectrum <= 0., 0.01, input\_spectrum))

含义是当input\_spectrum<=0时返回0.01,其他不为0时返回本身

#### 寻找最值索引argmax/argmin

返回某个向量的最大最小值所在的位置索引

#### 向量数值统计bincount

可以对一维数组所有值出现的个数进行统计，已经排序，可用于寻找图像像素强度的分布

#### 累加cumsum

cumsum(a,axis)

指定按axis轴累加

## Matplotib库

### 面向对象

#### 多子图subplots

plt.figure(dpi=600,figsize=(8,4))

fig,axes = plt.subplots(2,5,sharex="row") # 支持row,col,all

axes = axes.flatten()

for idx,axe in enumerate(axes) :

pack\_params = dict(cmap=plt.cm.Greys,norm=None,aspect=None,interpolation='nearest',clim=(0,255))

axe.axis("off")

if idx < 5 :

axe.imshow(x\_train[idx,:],\*\*pack\_params)

else:

axe.imshow(X\_train\_pca\_inverse[idx-5, :], \*\*pack\_params)

plt.show()

#### 设置轴axis

axe.axis("off")

#### 设置轴名set\_xlabel/set\_ylabel

#### 设置刻度set\_xticks/set\_yticks

#### 竖直线axvline

针对轴axe的方法

#### 双y轴twinx

fig , ax = plt.subplots()

axx = ax.twinx()

ax.plot(Train\_loss,label='loss',color='b')

axx.plot(Acu,label='acu',color='r')

fig.legend()

fig.show()

#### 三维Axes3D

from mpl\_toolkits.mplot3d import Axes3D

fig = plt.figure()

ax = Axes3D(fig)

X,Y,Z=test\_encoder.data.numpy()[:,0],test\_encoder.data.numpy()[:,1],test\_encoder.data.numpy()[:,2]

ax.set\_xlim([min(X),max(X)])

ax.set\_ylim([min(Y),max(Y)])

ax.set\_zlim([min(Z),max(Z)])

plt.show()

#### 添加子图add\_subplot

fig = plt.figure()

fig.add\_subplot()

#### 图表字段参数

dict(title="title",legend=["sin","cos"],dash=np.array(["solid","dash","dashdot"]),markersize=4,xlabel="feature1",ylabel="feature2",rownames=["x1","x2"],colnames=["x1","x2"])

### 面向过程

#### 画布figure

f=plt.figure()

ax=f.add\_subplot(1,1,1)

#### 条形图bar

# 可选颜色,标记,线型见

https://blog.csdn.net/syyyy712/article/details/87426927

# 颜色类型见

https://stackoverflow.com/questions/22408237/named-colors-in-matplotlib

color =

['red','green','blue', 'pink', 'orange', 'yellow','gold','skyblue','cyan','brown']

reats =

plt.bar(maxminVotes[0], maxminVotes[3], width=0.35, align='center', color=color, alpha=0.8)

for reat in reats: # 每一个柱子循环标注数值大小

reat\_height = reat.get\_height() # 获取柱子高度

plt.text(reat.get\_x() + reat.get\_width() / 2, reat\_height, str(reat\_height), size=10.5, ha='center',va='bottom')

plt.tight\_layout()

plt.xticks(ticks=np.arange(0, len(maxminVotes[0])))

plt.title("confidence for vote")

plt.yticks(ticks=np.arange(0,1.1,0.1)) # Y同理改变轴刻度为0,10,20,...,100

plt.xlabel("label")

plt.ylabel("confidence")

plt.show()

#### 颜色棒colorbar

plt.colorbar(ticks=range(10),label='digit value')

Ticks规定了颜色的刻度范围

Label给颜色棒加个标题

#### 绘制垂线vlines/hlines

for idx , value in enumerate(cumsum) :

if value >=0.9 :

x0 = idx

y0 = value

break

plt.plot(cumsum),plt.vlines(x0,ymin=0, ymax=y0),

plt.hlines(y=y0, xmin=0, xmax=x0),plt.show()

#### 子图subplot

相关用法：

for idx,(flower\_name,dir\_path) in enumerate(zip(flower\_names,dir\_paths)):

img = plt.imread(dir\_path+"\\"+os.listdir(dir\_path)[0]) # 第一张图片

plt.subplot(2,3,idx+1)

plt.title(flower\_name)

plt.axis('off')

plt.imshow(img)

plt.show()

还可以使用：

plt.subplot(121)

...

plt.subplot(122)

## Seaborn库

### 热图heatmap

sns.heatmap(df\_cm,annot=True,annot\_kws={"size":16},fmt=".3g",cbar=True)

sns.heatmap(data,vmin=None,vmax=None,cmap=None,center=None,robust=False,annot=None,fmt=".2g",annot\_kws=None,linewidths=0,linecolor="white",cbar=True,cbar\_kws=None,cbar\_ax=None,square=False,xticklabels="auto",yticklabels="auto",mask=None, ax=None,)

data：可以强制转换为 ndarray 的二维数据集。如果提供了Pandas DataFrame，索引/列信息将用于标记列和行。

vmin：锚定颜色图的值，否则它们是从数据和其他关键字参数推断出来的

vmax：锚定颜色图的值，否则它们是从数据和其他关键字参数推断出来的

cmap：从数据值到颜色空间的映射。 如果未提供，默认值将取决于是否设置了center

center：绘制发散数据时将颜色图center的值。 如果未指定，则使用此参数将更改默认的cmap

robust：如果 True且vmin或vmax不存在，则使用健壮的分位数而不是极值来计算颜色图范围

annot：如果为 True，则在每个单元格中写入数据值

fmt：添加注释时使用的字符串格式化代码

annot\_kws：用于设置单元格格式的一些参数

linewidths：将划分每个单元格的线的宽度

linecolor：将划分每个单元格的线条的颜色

cbar：是否绘制cbar

cbar\_kws：:设置颜色条的关键字参数

cbar\_ax：绘制cbar\_ax轴，否则占用主轴的空间

squar：如果为True，将 Axes 方面设置为相等，以便每个单元格都为方形

xticklabels：如果为True，则绘制数据xticklabels的列名称

### 设置风格set\_style

## pandas库

### Series

pd.Series(data=None,index=None,dtype=None,name=None,copy = False,fastpath=False)

data：可以使用列表

index：可以使用列表、数组：['0','1','2']或range(3)，指定索引

可以使用dict来初始化值，那么key成为index，value成为data

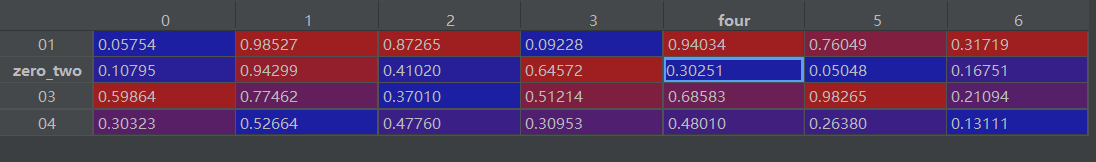
Series['0']访问的是指定行的值

### DataFrame

DataFrame( data=None,index=None,columns= None,dtype= None,copy= None,)

同样可以使用字典来指定data和columns，这里不同的dict的key用来指定的不是index，而是columns

DataFrame['key']访问的某列



Xy是一个末尾列为label的表格,以下代码可以找到label=0的数据

class0 = Xy[Xy["label"]==0]

### 设置精度set\_option

pd.set\_option('precision',2,'max\_columns',2,'display.width',None)

设置显示精度、输出时每行显示的最大列数（便于窗口显示）

### 内置函数

#### 按key/value排序sort\_index/sort\_values

dat.sort\_index(axis=0,ascending=False)

按行和升序排列，默认按首字母排序

一般是对值排序，则使用

dat.sort\_values(by='0',axis=1,ascending=False)

by指定排列的关键标签索引，axis指定排列轴

#### 重设行名和列名

重设行标签

dat.index = ['01','02','03','04']

重设列标签

dat.columns = ['0','1','2','3','4','5','6']

#### 更名rename

dat.rename(columns={"4":"four"},index={"02":"zero\_two"},inplace=True)

inplace必须为True，否则不改变原有值

#### 索引loc/iloc

loc需要给出['index','columns']，字符串参数

dat.loc['zero\_two','four'] = 0.30251

iloc为整数参数，即索引位置

dat.iloc[1,4] = 0.30251

类似的可用at和iat来访问和更改特定单元格

#### 采样sample

dat.sample(n=None,frac=None,replace=False,weights=None,random\_state=None,axis= None,ignore\_index= False,)

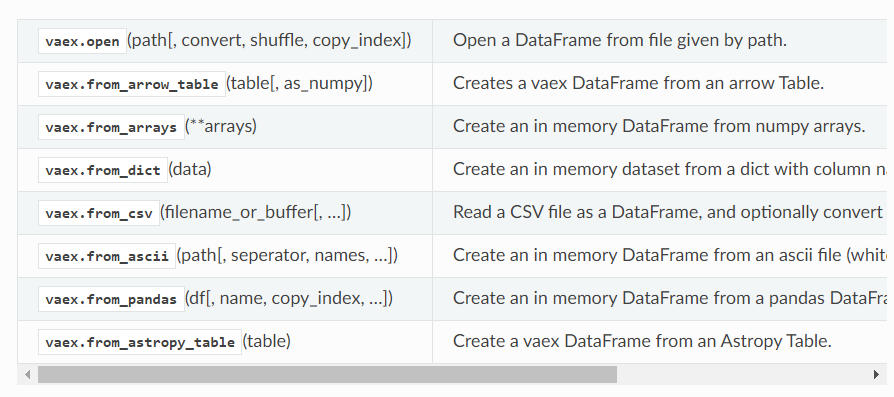
抽样几个指定n参数，抽样比例指定frac，二选一

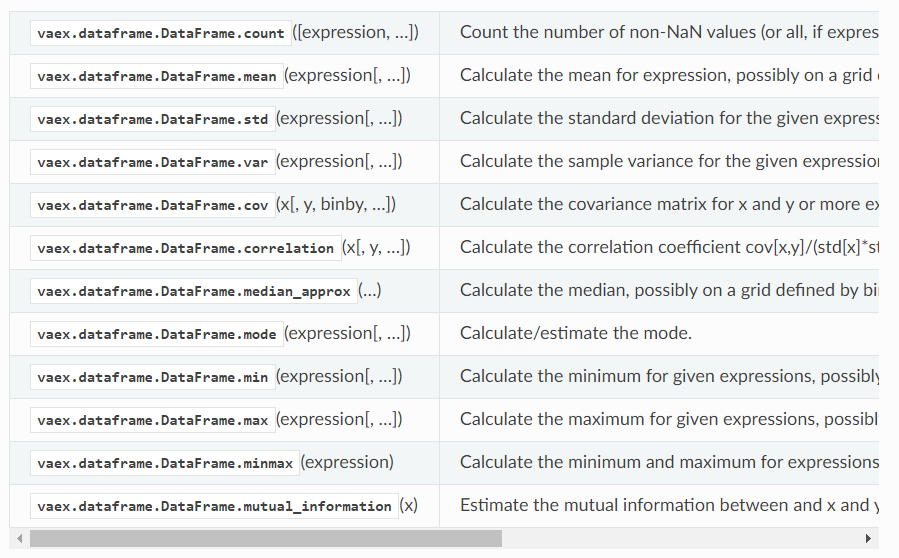
replace：允许或不允许对同一行进行多次采样，也就是是否放回抽样

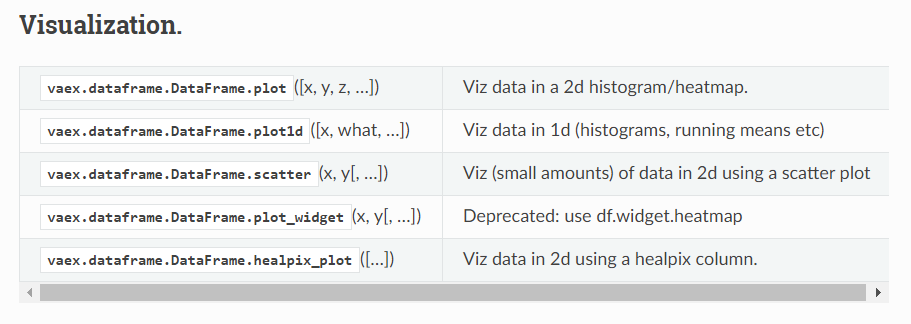
ignore\_index：忽略索引，如果希望知道位置来做其他操作，使用False，否者返回的索引为0, 1, …, n - 1

## Vaex库

pip install vaex







### 设置主题风格set\_theme

# context – 缩放参数

# style – 轴样式参数

# palette – 调色板

# font – 字体系列

# font\_scale – 单独的缩放因子以独立缩放字体元素的大小

# rc – rc 参数映射字典以覆盖上述内容

# set\_theme和set函数是一样的

sns.set\_theme(context="notebook", style="darkgrid", palette="deep",

font="sans-serif", font\_scale=1, color\_codes=True, rc=None)

sns.set\_style("whitegrid") # 默认白色背景

sns.set(font\_scale=1.2) # 字体

### 绘制热图heatmap

# annot使用fmt和annot\_kws的格式注释

# center = 0 使用发散颜色图绘制以0为中心的数据的热图

# 颜色棒用cbar控制，范围vmin和vmax控制

# linecolor='r',linewidths=1,控制表格线宽和颜色

# xticklabels=2, yticklabels=False 控制表格的刻度

uniform\_data = np.random.randn(12, 10)

heatmap=sns.heatmap(uniform\_data,center=0,annot=True,annot\_kws={"size":10},fmt=".3g",cbar=True,linecolor='r',linewidths=1,vmin=0,vmax=1,xticklabels=2,yticklabels=False)

# 设置刻度和标签

plt.show()

常常与混淆矩阵配合使用：

cm = confusion\_matrix(y\_test,LR.predict(X\_test))

cm = pd.DataFrame(cm)

sns.set\_style("whitegrid") # 默认白色背景

sns.set(font\_scale=1.2) # 字体

sns.heatmap(cm,annot=True,annot\_kws={"size":16},fmt="d",cbar=True) # 单元格写入数值和绘制颜色条

## Markdown语法

### 斜体\* \* Ctrl+b

### 加粗\*\* \*\* Ctrl+i

### 一级标题 ## ctrl+1

### 二级标题 ### ctrl+2

### 下划线<u></u> Ctrl+u

### 流程图mermaid

```mermaid

Graph TD (top->down)

A[开始]

B[结束]

A🡪|指向|B

# 图像处理工具

## Pillow库

### Image模块

#### 打开图像open

open()

#### 转换图像格式convert

convert()

“1”、“L”、“P”“RGBA”和“RGB”

分别对应二值、灰度、调色板、彩色、彩色透明度图像

#### 裁剪图像crop

crop(box)

坐标系向下y，向右x，box是1个元组，(left, upper, width, height)

表示截取的区域到左边、上方的距离，以及区域的size

#### 调整图像尺寸resize

resize(size,resample)

resample定义图像采样方式：

PIL.Image.BILINEAR

PIL.Image.BICUBIC

PIL.Image.HAMMING

PIL.Image.LANCZOS

IL.Image.NEAREST

#### 图像点处理point

point(lut)

lut可以使用lambda函数，对每个点进行处理

#### 旋转图像rotate

rotate(angle)

#### 转置图像transpose

transpose()

#### 变换图像transform

transform(size,method,resample)

PIL.Image.EXTENT（切出一个矩形子区域）

PIL.Image.AFFINE（仿射变换）

PIL.Image.PERSPECTIVE（ 透视变换）

PIL.Image.QUAD`（将四边形映射到矩形）

PIL.Image.MESH`（在一次操作中映射多个源四边形）

#### 拷贝图像copy

copy()

#### 分离图像通道spilit

spilit()

#### 融合图像blend

blend(im1,im2,alpha)

out = image1 \* (1.0 - alpha) + image2 \* alpha

#### 图像滤波filter

filter(filter)

其中filter为ImageFilter模块中指定的滤波器类型

### ImageFilter模块

一般需要配合filter(filter)函数使用

#### 有参数滤波器

#### 最大值滤波器maxFilter

ImageFilter.MaxFilter(size)

#### 最小值滤波器minFilter

ImageFilter.MinFilter(size)

#### 中值滤波器medianFilter

ImageFilter.MedianFilter (size)

#### 无参数滤波器

##### BLUR滤波器

ImageFilter.BLUR

##### FIND\_EDGES滤波器

ImageFilter.FIND\_EDGES

##### EDGE\_ENHANCE滤波器

ImageFilter. EDGE\_ENHANCE

##### EDGE\_ENHANCE\_MORE滤波器

ImageFilter.EDGE\_ENHANCE\_MORE

### ImageEnhance模块

#### 调整对比度Contrast

Contrast()

Eg：需事先生成一个类，再使用内置函数enhance()增强对比度

imgContrast = ImageEnhance.Contrast()

imgCore = imgContrast.enhance().getdata()

### ImageDraw模块

#### 绘制椭圆ellipse

# xy 的含义与crop一致 后2个表示椭圆宽度直径和高度直径 fill表示颜色

draw\_ellipse.ellipse(xy=(0,0,200,300), fill=(0,255,0))

#### 绘制文本text

draw\_font = ImageFont.truetype(font="arial.ttf", size=23)

# 其中draw\_font可以ImageFont来定义

draw\_text.text(xy = (0,0),text="chenbei",font=draw\_font)

#### 绘制矩形框rectangle

draw\_rectangle = ImageDraw.Draw(img)

draw\_rectangle.rectangle(xy=[100,0,300,100],fill=None,outline="green")

# （X，Y，W，H）格式

img.show()

## Skimage库

pip install scikit-image==0.17.2 --user

### Color模块

图像格式转换模块

#### rgb2gray

彩色图变为灰度图

#### label2rgb

标签并不是像素强度，还需要根据标签转换，给定参数标签和被转换对象

将指定标签区域的图像转换为rgb格式，可见图像区域划分label的使用，即API scipy.ndimage.label()

### Draw模块

#### 生成圆周长坐标circle\_perimeter

指定圆中心和半径，一般是二维图像的中心位置，返回以中心位置半径的圆，点的密集度取决于半径， 8倍关系，这说明每扩大一个半径增加8个采样点

import skimage.draw

im = np.zeros((10,10)),radius = 3

rr,cc =

skimage.draw.circle\_perimeter(r=im.shape[0]//2,c=im.shape[1]//2,radius=radius)

pixels = np.vstack((rr,cc)).T

plt.imshow(im)

for pixel in pixels :

plt.scatter(pixel[0],pixel[1])

plt.show()

# 点的密集度测试

for radius in [1,2,3,4,5,6,7] :

a,b = skimage.draw.circle\_perimeter(5,5,radius=radius)

print(len(a),len(b)) # 8倍关系，这说明每扩大一个半径增加8个采样点

### Expousure模块

#### 调整对比度rescale\_intensity

rescale\_intensity(image, in\_range='image', out\_range='dtype')

默认输入强度范围使用输入图像本身的强度范围，输出为输入图像类型应有的范围，in\_range可以使用字符串'image'、'dtype'，也可以使用uint10、'uint12'、'uint14'、'bool'、'float'，还可以使用二元组2-tuple()

举例、

1. 输入是无符号整数uint8，[51, 102, 153] ，那么输出默认会调整到应有的uint8的范围，即[ 0, 127, 255]
2. 输入是浮点类型float，[ 51., 102., 153.]，那么输出默认会调整到float具有的范围，即[0. , 0.5, 1. ]，但是标准是153为1，51为0
3. 如果希望仍然是以255为标准去调整，就要指定in\_range=(0,255),输出此时为[0.2, 0.4, 0.6]，即153/255=0.6
4. 如果希望基准线不仅不是255，而且调低也可以，指定in\_range=(0,102),那么输出为[0.5, 1. , 1. ]，凡是大于102的都为1.
5. 如果输入是有符号整数，[-10, 0, 10]，希望输出限制在127以内，即out\_range=(0,127)，那么输出为[ 0. , 63.5, 127. ]
6. 输入是常数uint8，[130, 130, 130]，也可以限制，out\_range=(0,127)，输出为[127, 127, 127]

#### 全局直方图均衡equalize\_hist

#### 自适应局部直方图均衡equalize\_adaphist

#### 累积分布系数cumulative\_distribution

### Feature模块

图像特征提取模块

#### 斑点检测器

##### 基于高斯拉普拉斯LoG

blob\_log()

##### 基于高斯差分DoG

blob\_dog()

##### 基于黑塞矩阵DoH

blob\_doh()

##### 基于方向梯度直方图HoG

blob\_hog()

#### 边缘检测器canny

image = canny(image)

#### 哈里斯角点检测器haars

##### 角点坐标匹配

corner\_harris()

##### 角点峰值计算

corner\_peaks()

可使用peak\_local\_max()替代

##### 细化角点峰值

corner\_subpix()

#### 基于特征符图像匹配

##### 基于短二进制特征描述符BRIEF

##### 基于BRIEF改进版ORB

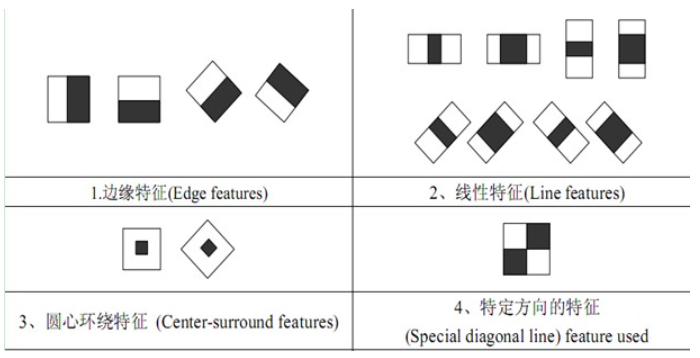
##### 匹配器match\_descriptors

##### 绘制匹配线plot\_matches

#### 类haar特征描述符

##### 计算类Haar特征的坐标haar\_like\_feature\_coord

Haar特征大致分为4种类型：边缘特征、线特征环、中心环绕特征和对角线特征，组合成特征模板，特征模板内有白色和黑色两种矩形。



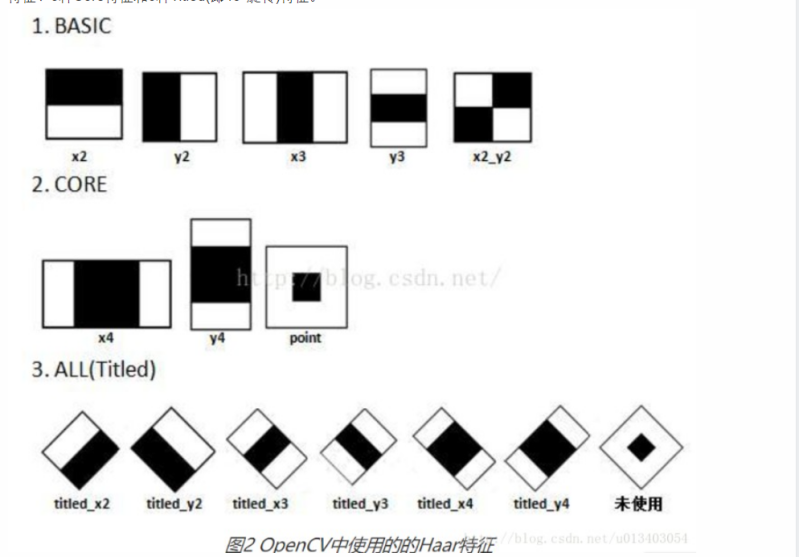
特征值的计算过程中，黑色区域的权值为负值，白色区域的权值为正值。而且权值与矩形面积成反比（使两种矩形区域中像素数目一致）

在使用opencv自带的训练工具进行训练时，haarFeatureParams参数中的mode参数正对应了训练过程当中所使用的特征集合。

1. 若是mode为BASIC，则只使用BASIC的5种Haar特征进行训练，训练出的分类器也只包含这5种特征。

2. 若是mode为CORE，则使用BASIC的5种+CORE的3种Haar特征进行训练。

3. 若是mode为ALL，则使用BASICA的5种+CORE的3种+ALL的6种Titled共14种特征进行训练。



特征值计算公式=全部像素和\*权重all + 黑色区域像素和\*权重black

对于图中的x3和y3特征，weightall = 1，weightblack = -3；

对于point特征，weightall = 1，weightblack = -9；

其他11种特征均为weightall =1， weightblack = -2。



feature\_coord ,feature\_type = haar\_like\_feature\_coord

(width=3,height=3,feature\_type='type-3-y')

##### 类Haar特征的可视化draw\_haar\_like\_feature

haar\_feature = draw\_haar\_like\_feature(image=im, r=0, c=0, width=3, height=3, feature\_coord=feature\_coord,max\_n\_features=1,alpha=0.8)

### Filters模块

#### 计算二值图像最佳阈值ostu

ostu()，返回最佳的二值图像分割阈值

#### 边缘检测器

##### 基于laplace

##### 基于sobel

sobel\_h()

sobel\_v()

sobel()

##### 基于roberts

roberts\_pos\_diag()

roberts\_neg\_diag()

##### 基于prewitt

prewitt\_v()

prewitt\_h()

##### 基于scharr

scharr\_h()

scharr\_v()

#### rank模块形态学计算

##### 调整对比度enhance\_contrast

##### 中值滤波median

##### 计算局部熵entropy

### Measure模块

#### 随机抽样一致性检验ransac

### Metrics模块

#### 峰值信噪比peak\_signal\_noise\_ratio

### Morphology模块

本模块主要用于形态学处理

#### 扁平矩形rectangle

rectangle(nrows=2,ncols=5)

生成全1的矩形元素

#### 盘状结构disk

disk(radius=4)

生成盘状结构的元素，指定半径即可，半径内元素均为1

#### 方形结构square

square(width)

#### 腐蚀binary\_erosion

binary\_erosion(image,selem)

腐蚀就是将暗的区域扩大，例如十字架核移动的时候，处于十字架的像素如果有1就将其改为0，膨胀正好相反。至于核形状核大小的选择取决于实际情况

提取边界的运算

im\_boundary = im - im\_erosion

#### 膨胀binary\_dilation

膨胀是将亮的区域扩大

#### 开运算binary\_opening

binary\_opening(image,selem)

开运算:先进行腐蚀再进行膨胀，可以消除离散点和毛刺（如椒盐噪声），可以将两个物体分开

#### 闭运算binary\_closing

binary\_closing(image,selem)

闭运算:先进行膨胀再进行腐蚀，可以填充图像的内部孔洞和图像的凹角点，把两个邻近的目标连接起来

#### 骨架化skeletonize

Skeletonize(image)

细化用于将二值图像中的每个连接组件减少到单像素宽的骨架,常用于模式识别

#### 凸包运算convex\_hull\_image

convex\_hull\_image(image)

物体的凸包检测常应用在物体识别、手势识别及边界检测等领域,类似于骨架化，可以得到突出的图像特征，然后用于检测

二维欧几里得空间中，凸包可想象为一条刚好包住所有点的橡皮圈

生成一个凸多边形，这个凸多边形将图片中所有的白色像素点都包含在内的运算,图像示例如下，左边是原图，右边是生成的凸多边形将内部包围



#### 删除小对象remove\_small\_objects

remove\_small\_objects(ar, min\_size=64)

可以用于移除二进制未填充的轮廓，需要设定小对象的最小尺寸，可以删除图像的孤立像素点

#### 白顶帽运算white\_tophat

white\_tophat(image,selem)

图像的白顶帽运算定义为图像减去其形态学开运算,得到图像的噪声



#### 黑顶帽运算black\_tophat

black\_tophat(image,selem)

图像的黑顶帽运算定义为形态学闭运算减去图像，得到图像内部的小孔，或者前景色中的小黑点



#### 形态学分水岭算法watershed

① 计算图像对比度或者梯度,可以使用sobel梯度

② 寻找原图像的标记markers，依据直方图极值划分做标记，极值为高频率像素分割处

watershed(image, markers=None)

### Restoration模块

#### 无监督维也纳滤波unsupervised\_wiener

#### 双边滤波denoise\_bilateral

#### 计算高斯噪声标准差estimate\_sigma

#### 非局部均值去噪denoise\_nl\_means

#### 双调和方程修复inpaint\_biharmonic

# 基于已知的二进制掩膜D，这个D标定了图像的受损像素

# 使用双调和方程假设修复图像 inpaint\_biharmonic

inpaint.inpaint\_biharmonic(im\_bad,mask,multichannel=True)

#### TVD全变分去噪denoise\_tv\_chambolle

去噪权重越大去噪效果越好，但是会牺牲图像的保真度从而导致图像模糊

denoise\_tv\_chambolle(im\_noise,weight=1)

### Segmentation模块

#### 获取初始水平集checkerboard\_level\_set

#### 高斯梯度反变换inverse\_gaussian\_gradient

#### 典型图像分割算法

##### 基于费尔森茨瓦布felsenszwalb

##### 基于k均值聚类slic

##### 基于快速移位quickShift

##### 基于紧凑型分水岭watershed

##### 基于活动轮廓学蛇active\_contour

##### 基于形态学测地线活动轮廓morphgac

##### 基于形态学无边缘活动轮廓morghacwe

### Transform模块

图像变换模块

#### 高斯金字塔pyramid\_gaussian

#### 拉普拉斯金字塔pyramid\_laplacian

#### 金字塔向下采样平滑pyramid\_reduce

#### 金字塔平滑向上采样pyramid\_expand

#### 调整图片尺寸rescale

#### 旋流变换swirl

#### 几何变换warp

配合一些几何变换使用，如相似变换、仿射变换等

#### 霍夫变换hough

用于检测线和圆

##### 线变换hough\_line

##### 线变换峰值hough\_line\_peaks

##### 圆变换hough\_circle

##### 圆变换峰值hough\_circle\_peaks

#### 相似变换SimilarityTransform

#### 仿射变换AffineTransform

### Util模块

#### 高斯随机噪声random\_noise

image = random\_noise(image,seed=random\_state,clip=True,mean=0,var=1) # 添加高斯噪声

## Scipy库

### Signal模块

#### 一维高斯分布gaussian

#### 二维卷积convolve2d

对RGB或者RGBA图像需要分通道进行卷积

#### 二维相关correlate2d

### FFTpack模块

#### 二维傅里叶变换fft2

#### 二维傅里叶逆变换ifft2

#### 零频率分量中心化fftshift

针对频率图使用

#### 零频率分量四周化ifftshift

针对时域图使用

### Ndimage模块

#### 卷积convolve

无需分通道，可直接对图像卷积

#### 傅里叶高斯核频谱fourier\_gaussian

#### 傅里叶盒核频谱fourier\_uniform

#### 傅里叶椭球核频谱fourier\_ellipsoid

#### 高斯核滤波gaussian\_filter

#### 盒核滤波uniform\_filter

#### 多维百分位滤波percentile\_filter

#### LoG滤波gaussian\_laplace

#### 图像区域划分label

输入二值图像input中的任何非零值都被视为特征，零值被视为背景

理解为重新调整分布，原来的特征是1，但是1的分布不是连续的，将这些1继续细分多个区域，改变了像素亮度

label分成了num\_features个特征+1个背景

num\_features 为细分后的区域种类

标签并不是像素强度，还需要根据标签转换，给定参数标签和被转换对象，可参考label2rgb的使用

label , num\_features = scipy.ndimage.label(input=seg\_)

#### 形态学计算

##### 二进制填充binary\_fill\_holes

可以用于填充轮廓，一般事先使用边缘检测器检测边缘，之后对边缘填充即可。

##### 开运算grey\_opening

##### 闭运算grey\_closing

##### 膨胀grey\_dilation

##### 腐蚀grey\_erosion

##### 多维形态梯度morphological\_gradient

##### 多维形态拉普拉斯morphological\_laplace

参考cv2的方法，形态学方法morphologyEx

## CV2(Python)

### 安装python平台

pip install opencv-python -i https://pypi.tuna.tsinghua.edu.cn/simple（下载cv2的命令）

pip install opencv-contrib-python --user # 使用管理员命令下载

### 等待按键waitKey

im = cv2.imread(img\_dir+'\lena.jpg',flags=1) # CV2使用的是BGR通道而非RGB

im\_ = cv2.cvtColor(src=im,code=cv2.COLOR\_RGB2GRAY)

cv2.imshow('lena',im\_)

# 默认等待无限长，ret为与按下键对应的ASCII码，如ESC对应27若没有则为-1

ret = cv2.waitKey(5000) # ms

cv2.destroyAllWindows()

# cv2.imwrite(filename=img\_dir, img=im) # 保存

### 命名窗口namedWindow

namedWindow(winname[, flags]) -> None

### 调整窗口大小resizeWindow

cv2.resizeWindow(winname, width, height) -> None

### 图像格式转换cvtColor

cv2.cvtColor(src=im,code=cv2.COLOR\_BGR2GRAY)

cv2.COLOR\_RGB2GRAY

### 分离通道spilit

B , G , R = cv2.spilit(im)

### 合并通道merge

Im = cv2.merge([B,G,R])

### 内范围inRange

C++中为inRange(InputArray src, InputArray lowerb,

InputArray upperb, OutputArray dst)

这里不必指定dst，lowerb是数组，upperb也是数组，需要指定每个通道的阈值，低于阈值和高于阈值的范围像素为0，阈值之间的为255

Mat src = imread(imgAddr+"apple.jpg"),hsv,dst;

cvtColor(src, hsv, COLOR\_BGR2HSV);

Vec3i lower = Vec3i(10, 10, 10);

Vec3i upper = Vec3i(200, 200, 200);

inRange(hsv,lower,upper,dst);

python中就较为简单，数组只需要使用numpy指定即可

### 绘制图形

#### 线段line

指定颜色、线宽以及线段起始点和终点，不需要指定通道，其中点坐标系为（x，y），不是数组索引，表示到左边的距离x和到上边的距离y

cv2.line(img=im,pt1=(50,50),pt2=(100,100),color=(0,0,255),thickness=20)

#### 矩形rectangle

可以绘制矩形边框也可以绘制实心矩形

pt1对应矩形左上角坐标，pt2为右下角坐标，当thickness=-1绘制的为实心矩形；

cv2.rectangle(img=im,pt1=(50,10),pt2=(100,180),color=(0,0,255),thickness=10)

注意指定的是数组像素坐标（x，y），也就是对应数组坐标是（raw，col）=（10，50），即x=50，高度y=10

#### 圆形circle

使用的坐标系相同，直接指定像素的坐标，不是数组索引

cv2.circle(img=im,center=(50,100),radius=10,color=(0,0,255),thickness=10)

#### 多边形polylines

pts为多边形顶点组成的列表，先使用array再列表化

lst = [np.array([[50,50],[150,50],[200,100],[0,100]])]

im = cv2.polylines(img=im,pts=lst,isClosed=True,color=(0,0,255),thickness=1)

#### 文字putText

控制文字的内容、字体、大小、颜色、宽度、线型和方向，org为文字的左下角坐标，bottomLeftOrigin默认False，lineType默认8，只有4和8两个选项，fontFace字体样式有斜体、手写风格等

kw = {'text':'cb','org':(20,150),'fontFace':cv2.FONT\_ITALIC,'fontScale':2,

'color':[0,255,0],'thickness':2,'lineType':8,'bottomLeftOrigin':False}

cv2.putText(img=im,\*\*kw)

关于文字风格：

cv2.FONT\_ITALIC # 斜体

cv2.FONT\_HERSHEY\_SIMPLEX # 正常大小sans-serif字体

cv2.FONT\_HERSHEY\_PLAIN # 小号的sans-serif字体

cv2.FONT\_HERSHEY\_DUPLEX # 正常大小sans-serif字体，比SIMPLEX复杂

cv2.FONT\_HERSHEY\_COMPLEX # 正常大小serif字体

cv2.FONT\_HERSHEY\_COMPLEX\_SMALL

cv2.FONT\_HERSHEY\_TRIPLEX # 正常大小serif字体，比CONMPLEX复杂 垂直镜像效果

cv2.FONT\_HERSHEY\_SCRIPT\_SIMPLEX # 手写风格字体

cv2.FONT\_HERSHEY\_SCRIPT\_COMPLEX

### 阈值处理

#### 普通处理threshold

Src：被处理的图像

Thresh：阈值

Maxval：阈值使用的最大值

Type：阈值处理类型

Retval：返回处理使用的阈值，也就是thresh

Dst：阈值处理后的图像

retval,dst=cv2.threshold(src=im,thresh=127,maxval=255,type=cv2.THRESH\_BINARY+cv2.THRESH\_OTSU)

cv2.THRESH\_BINARY # 二值化阈值，默认高于阈值为255，反之为0

cv2.THRESH\_BINARY\_INV # 反二值化阈值，大于阈值为0，反之255

cv2.THRESH\_TOZERO # 低于阈值零处理，高于阈值不变

cv2.THRESH\_TOZERO\_INV # 超出阈值零处理，低于阈值不变

cv2.THRESH\_TRUNC # 截断阈值，小于阈值必变，超过阈值等于阈值

cv2.THRESH\_OTSU # 寻找最优阈值，此方法类似于skimage.filters模块的函数计算二值图像最佳阈值

#### 自适应阈值处理adaptiveThreshold

色彩不均衡的图像，一般而言截断阈值处理是效果最好的，但是仍然不是最佳，本函数可以在不同的区域使用不同的阈值，能够更好的处理明暗不均的图像

adaptiveThreshold(src, maxValue, adaptiveMethod, thresholdType, blockSize, C)

Src：必须是灰度图像

Maxvalue：阈值处理最大值

AdaptiveMethod：自适应方法

① 对一个正方形区域内的所有像素平均加权

cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C

② 根据高斯函数按照像素点到中心距离对一个正方形区域的像素加权平均

cv2.ADAPTIVE\_THRESH\_MEAN\_C

ThresholdType：阈值处理类型，必须是cv2.THRESH\_BINARY或者cv2.THRESH\_BINARY\_INV

Blocksize：正方形区域的大小

C：常量，阈值等于均值或者加权值减去该常量

### 几何变换

#### 缩放图像resize

resize(src, dsize, dst, fx, fy, interpolation)

Dsize：输出的尺寸，格式（宽，高）

Fx：可选，水平缩放比例，新图像宽度取整处理

Fy：垂直缩放比例，新图像高度取整处理

Interpolation：缩放插值方式

使用dsize或者使用fx、fy来缩放，此时dsize必须等于None

1. resize(im,(100,100))
2. resize(im,dsize=None,fx=1/3,fy=1/2)

#### 翻转图像flip

flip(src,flipCode)

flipCode=0，沿X轴翻转，=正数沿Y轴翻转，=负数，均翻转

#### 仿射变换warpAffine

仿射变换 = 平移 + 旋转 + 倾斜

warpAffine(src, M, dsize, flags, borderMode, borderValue)

M：1个2行3列的仿射矩阵，根据此矩阵值变换原图像素，矩阵数字格式float32

原像素为[x,y]，新像素[x1,y1]

对于平移，M的系数a、e均为1，d、b均为0即可，此时为平移变换

旋转，M也可以带入相应的系数，计算此旋转矩阵系数可以使用方法计算旋转矩阵getRotationMatrix2D

倾斜图像，使用计算倾斜矩阵getAffineTransform

Dsize：输出图像尺寸大小

Flags：可选，插值方式

borderMode：边界类型

BorderValue：边界值，默认0

#### 计算旋转矩阵getRotationMatrix2D

getRotationMatrix2D(center, angle, scale)，需要指定旋转的中心点坐标、旋转角度（不是弧度，正数为逆时针旋转）、缩放比例（取值1.0表示保持不变）

center = (rows/2,cols/2) # 一般取图像中心点

#### 计算倾斜矩阵getAffineTransform

getAffineTransform(src, dst)

src为3行2列的列表，即原图像的左上角、右上角和左下角的坐标

dst为倾斜后的图像左上角、右上角和左下角的坐标

P1,P2 = np.zeros((3,2),np.float32),np.zeros((3,2),np.float32)

P1[0] = [0,0] # 左上角原点

P1[1] = [cols-1,0] # cols为图像数组的列数，实际上是图像的宽度，但是坐标应该不从0开始，所以还要减去1，这样等于右上角的坐标

P1[1] =[0,raws-1] # rows为数组行数也就是高度

P2[0] = [50,0] # 右倾斜50像素

P2[1] = [cols-1,0] # 不变

P2[2] = [0,rows-1] # 不变

M = getAffineTransform(P1,P2)

Im\_re = warpAffine(im,M,(cols,rows))

如果需要左倾斜，不同于右倾斜只需要改变左上角坐标，这里需要改变右上角向左50像素，左下角向右50像素

#### 透视变换warpPerspective

warpPerspective(src, M, dsize, dst, flags, borderMode, borderValue)

M为透视矩阵，可以由计算透视矩阵getPerspectiveTransform计算，返回3×3的矩阵M

row , col = im.shape[0] , im.shape[1]

p1 , p2 = np.zeros((4,2),np.float32) , np.zeros((4,2),np.float32)

# 原图4个顶点坐标位置

p1[0],p1[1],p1[2],p1[3] = [0,0],[col-1,0],[0,row-1],[col-1,row-1]

# 透视图图4个顶点坐标位置，左上角右移，右上角左移

p2[0],p2[1],p2[2],p2[3] = [90,0],[col-1-90,0],[0,row-1],[col-1,row-1]

dst=cv2.warpPerspective(src=im,M=cv2.getPerspectiveTransform(src=p1,dst=p2),dsize=im.shape,flags=None, borderMode=None, borderValue=None)

#### 计算透视矩阵getPerspectiveTransform

getPerspectiveTransform(src, dst)

带入原图和透视图后的4个顶点坐标

### 图像四则运算

#### 加减add

Opencv的三原色符合光学三原色，即红绿蓝=白色，颜料中的三原色为红黄蓝=黑色

add(src1, src2, mask, dtype)

mask为掩码，dtype为图像深度

# 比较+和add

im1=np.array([[0,0,0,0,0],[255,255,255,255,255],[0,0,0,0,0],[255,255,255,255,255],[0,0,0,0,0]],dtype=np.uint8)

im2=np.array([[0,255,0,255,0],[0,255,0,255,0],[0,255,0,255,0],[0,255,0,255,0],[0,255,0,255,0]],dtype=np.uint8)

mask=np.array([[0,0,0,0,0],[0,0,0,0,0],[0,0,0,0,0],[255,255,255,255,255],[255,255,255,255,255]],dtype=np.uint8)

dst1 = im1 + im2 # uint8型会对(255+255)取模得到254

dst2 = cv2.add(im1,im2) # 超出部分限制为255

dst3 = cv2.add(im1,im2,mask=mask)# 计算完后再取掩码，只保留下半部分

#### 加权和addWeighted

addWeighted(src1, alpha, src2, beta, gamma, dtype)

需要给出两幅图像的权重α和β，并给出和结果添加的亮度参数gamma，值越大越亮，可以是负数

#### 减法substract

subtract(src1, src2, dst, mask, dtype)

#### 乘法multiply

multiply(src1, src2, dst, scale, dtype)

#### 除法divide

divide(src1, src2, dst, scale, dtype)

### 位运算bitwise

位运算会将图像每个像素值转为二进制数，然后两个图像相同位置进行运算

位与：bitwise\_and(src1,src2,mask)

位或：bitwise\_or(src1,src2,mask)

位取反：bitwise\_not(src,mask)

位异或：bitwise\_xor(src,mask)

### 图像滤波

#### 均值滤波blur

blur(src, ksize, dst, anchor, borderType)

dst, anchor, borderType分别为输出的图像、滤波核锚点、边界样式使用默认参数即可，ksize=（高，宽）

#### 中值滤波medianBlur

medianBlur(src, ksize, dst)

适用于椒盐噪声、脉冲噪声，中值滤波器的ksize为变长，而不是（高，宽），也就是一般是正方形的

#### 高斯滤波GaussianBlur

GaussianBlur(src, ksize, sigmaX, dst, sigmaY, borderType)

需要给定卷积核水平核竖直方向的标准差sigmaX核sigmaY，可以都设为0，方法会根据滤波核的大小自动计算合适的权重

#### 双边滤波bilateralFilter

均值、中值核高斯滤波会使得图像平滑，但边缘模糊不清，双边滤波可以有效保护边界信息

bilateralFilter(src, d, sigmaColor, sigmaSpace, dst, borderType)

d：以当前像素为中心的整个滤波区域的直径，若d<0则根据sigmaSpace参数自动计算，该值与保留的边缘信息数量成正比，与方法运行效率成反比，可以看成中值滤波的ksize参数

sigmaSpace：坐标空间的sigma值，越大则参与计算的像素数量越多

### 形态学计算

#### 腐蚀erode

腐蚀可以向内收缩，消除外部细节

erode(src, kernel, dst, anchor, iterations, borderType, borderValue)

Kernel：腐蚀用的核，核参数的行列数越大计算的结果越粗糙

Iterations：腐蚀操作的迭代次数

#### 膨胀dilate

膨胀可以向外扩张，消除内部细节

dilate(src, kernel, dst, anchor, iterations, borderType, borderValue)

开运算 = 腐蚀 + 膨胀

闭运算 = 膨胀 + 腐蚀

#### 形态学方法morphologyEx

morphologyEx(src, op, kernel, dst, anchor, iterations, borderType, borderValue)

Op：操作类型，其他参数含义相同

cv2.MORPH\_ERODE

cv2.MORPH\_DILATE

cv2.MORPH\_CLOSE

cv2.MORPH\_OPEN

cv2.MORPH\_GRADIENT # 梯度运算=膨胀-腐蚀

梯度运算可以参考多维形态梯度morphological\_gradient

cv2.MORPH\_TOPHAT # 白顶帽=原始-开运算

cv2.MORPH\_BLACKHAT # 黑顶帽=闭运算-原始

#### 凸包convexHull

Hull = convexHull(points, hull, clockwise, returnPoints)

Points：轮廓数组

Clockwise：True表示凸包的点顺时针排列

ReturnPoints：True时返回点坐标，False返回索引，默认True

Hull：返回凸包的点阵数组

### 边缘检测

#### Canny检测

edges=Canny(image,threshold1,threshold2[,edges[, apertureSize[, L2gradient]]])

根据像素梯度的变换寻找边缘，使用双阈值

Aperturesize：sobel算子的孔径大小

L2gradient：图像梯度的标识，默认False，True会使用更精准的算法计算

Edges：返回二值灰度图像

#### 寻找轮廓findContours

contours,hierarchy=cv2.findContours(binary,mode=cv2.RETR\_TREE,method=cv2.CHAIN\_APPROX\_TC89\_KCOS)

# 被检测的图像必须是8位单通道二值图像

# mode为轮廓的检索模式

# cv2.RETR\_LIST 检测所有轮廓不建立层次关系

# cv2.RETR\_CCOMP 检测所有轮廓，建立两级层级关系

# cv2.RETR\_EXTERNAL 只检测外轮廓

# cv2.RETR\_TREE 检测所有轮廓，简历树状层级关系

# method为检测轮廓的方法

# cv2.CHAIN\_APPROX\_NONE 储存轮廓上的所有点

# cv2.CHAIN\_APPROX\_SIMPLE 只保存水平、垂直或对角线轮廓的端点

# cv2.CHAIN\_APPROX\_TC89\_L1 Ten\_Chinl近似算法的一种

# cv2.CHAIN\_APPROX\_TC89\_KCOS Ten\_Chinl近似算法的一种

# contours 返回list ，每个list都是一组像素点的坐标，变为array是向量

# hierarchy # 轮廓之间的层级关系

#### 绘制轮廓drawContours

im\_re=cv2.drawContours(image=im,contours=contours,contourIdx=len(contours)-1,color=[0,255,0],thickness=0,lineType=8,hierarchy=hierarchy,maxLevel=0,offset=None)

# image可以是多通道图像

# contours和hierarchy是findContours方法的返回值

# contourIdx绘制轮廓的索引，=-1表示全部绘制,索引小于contours的长度

# maxLevel 绘制轮廓的层次深度

# offset 偏移量，可以改变绘制结果的位置

#### 矩形包围框坐标boundingRect

#轮廓拟合矩形框需要找轮廓的水平垂直对角线端点

contours,hierarchy=cv2.findContours(binary,mode=cv2.RETR\_LIST,method=cv2.CHAIN\_APPROX\_SIMPLE)

# array 轮廓数组，一般是contours的某个list

# retval=x,y,w,h 依据轮廓数组的端点返回矩形的坐标返回元组

# 包含4个整数值，对应左上角的横、纵坐标以及矩形的宽和高

x,y,w,h = cv2.boundingRect(array=contours[-1])#最后1个轮廓

cv2.rectangle(im,pt1=(x,y),pt2=(x+w,y+h),color=[0,0,255],thickness=2)

#### 圆形包围框坐标minEnclosingCircle

center,radius = cv2.minEnclosingCircle(points)

# points 轮廓数组，一般是contours的某个list

# center 包含2个整数值，圆心坐标

# radius 半径

#### 霍夫检测线HoughLinesP

Lines=HoughLinesP(image,rho,theta,threshold,lines,minLineLength,maxLineGap)

Rho：检测直线使用的半径步长，值为1表示使用所有可能的半径步长

Theta：搜索直线的角度，值为pi/180表示检测所有角度

Threshold：阈值越小，检测的直线越多

minLineLength：线段最小距离，小于该长度的直线不被记录

MaxLineGap：线段间的最小距离

Lines：返回一个数组，元素为所有检测出的线段，每个线段也是数组，为线段2个端点的横纵坐标，格式为[[[x1,y1,x2,y2],[x3,y3,x4,y4]]]，即返回有多个直线的2个端点数组,第一维表示可能的直线数量

#### 霍夫检测圆HoughCircles

circles=cv2.HoughCircles(binary2,method=cv2.HOUGH\_GRADIENT,dp=1,minDist=70,param1=100,param2=25,minRadius=10,maxRadius=50)

circles = np.uint(np.round(circles)) # 像素坐标只能是整数

# method opencv4.0以前只提供唯一的方法cv2.HOUGH\_GRADIENT

# dp 累加器分辨率与原始图像分辨率之比的倒数，等于1时表示具有相同分辨率，为2时为原图分辨率的1/2

# minDist 圆心之间的最小距离

# param1 可选，Canny 边缘检测的最大阈值

# param2 可选，检测圆环结果的投票数，第一轮投票数超过该值的圆才会进入第二轮，越大检测的圆越少

# minRadius 圆最小半径

# maxRadius 圆最大半径

# circles 返回霍夫圆空间数组，每个数组内还是数组，记录了圆心坐标和半径，形如[[[x1,y1,r1],[x2,y2,r2]]],第一维表示可能的圆空间数量，第二维才是可能的圆数量，循环绘制每个圆时注意从第2维遍历

### 人脸检测

#### Eigenface人脸识别

model=cv2.face.EigenFaceRecognizer\_create(num\_components,threshold)

需要指定PCA的降维数和人脸识别的阈值，使用默认值

利用PCA的特征脸技术得到的人脸识别算法，特征脸是PCA转换矩阵的K个样本，可以参考主成分降维PCA

#### FisherFace人脸识别

cv2.face.FisherFaceRecognizer\_create(num\_components,threshold)

通过LDA线性判别分析得到的人脸识别算法，内置方法相同

#### LBPH人脸识别

Local Binary Pattern Histogram局部二值模式直方图算法，可以捕获局部纹理特征，内置方法相同

cv2.face.LBPHFaceRecognizer\_create(radius,neighbors,grid\_x,grid\_y,threshold)

Radius：圆形局部二进制模式半径

Neighbors：圆形局部二进制模式的采样点数目

Grid\_x：水平方向的单元格数

Grid\_y：竖直方向的单元格数

#### 内置方法训练train

model.train(src,labels)

src为图片样本列表list，高宽必须相同

labels为样本标签

#### 内置方法预测predict

label,confidence = model.predict(src)

label为预测的标签，confidence为置信度评分

### 模板匹配matchTemplate

模板是被查找的图像，查找模板在图像的哪个位置叫模板匹配

matchTemplate(image, templ, method[, result]) -> result

Templ：模板图像，尺寸小于原始图像

Method：匹配方法

# cv2.TM\_SQDIFF =0 平方差匹配 匹配程度越高计算结果越小 完全匹配的结果为0

# cv2.TM\_SQDIFF\_NORMED =1 标准平方差匹配

# cv2.TM\_CCORR =2 相关匹配，匹配程度越高计算结果越大

# cv2.TM\_CCORR\_NORMED =3 标准相关匹配

# cv2.TM\_CCOEFF =4 相关系数匹配 计算结果为-1~1之间的浮点数，1表示完全匹配，0表示毫无关系，-1表示亮度刚好相反

# cv2.TM\_CCOEFF\_NORMED # 标准相关系数匹配

Mask：可选，掩码，只对cv2.TM\_SQDIFF和cv2.TM\_CCORR\_NORMED支持

Result：返回匹配的结果，模板图像尺寸w、h，原始图像尺寸W、H，那么返回的结果为W-w+1列、H-h+1行的浮点数组，每个浮点数都是原始图像对应像素的匹配结果，含义由method解读

因为模板移动到边缘不会再移动，此时相当于只能横向移动W-w+1次，纵向移动H-h+1次

### 计算模板匹配坐标minMaxLoc

minMaxLoc(src[, mask]) -> minVal, maxVal, minLoc, maxLoc

Src为matchTemplate返回的数组，mask可选一般默认

返回的minVal为数组的最小值，maxVal最大值，对于平方差匹配找到minVal所处的索引坐标（x，y）minLoc，其他则是找到最大的坐标maxLoc

### 图像匹配算法

Matcher = cv2.ORB\_create()

#### SIFT\_create

#### ORB\_create

#### 内置方法

##### 计算关键点detect

##### 计算关键点和特征符detectAndCompute

kp1 , des1 = Matcher.detectAndCompute(im1,None)

### 匹配器BFMatcher

matcher = cv2.BFMatcher()

#### match

#### knnMatch

### 绘制匹配器

#### drawMatches

#### drawMatchesKnn

### 打开视频VideoCapture

VideoCapture() -> <VideoCapture object>

or VideoCapture(filename) -> <VideoCapture object>

or VideoCapture(device) -> <VideoCapture object>

打开内置摄像头

Capture = cv2.VideoCapture(index,cv2.CAP\_DSHOW) #打开速度快一些

打开指定视频

Video=cv2.VideoCapture(filename)

Eg：

capture\_inner = cv2.VideoCapture(0,cv2.CAP\_DSHOW) # index=0内置摄像头 =1外置 cv2.CAP\_DSHOW启动比较快

capture\_outer = cv2.VideoCapture(1,cv2.CAP\_DSHOW) # 打不开

while (capture\_inner.isOpened() or capture\_outer.isOpened()) :

retval\_inner , image\_inner = capture\_inner.read()

retval\_outer , image\_outer = capture\_outer.read()

#print(retval\_inner,retval\_outer)

if retval\_inner : # 如果读取到帧

cv2.imshow('inner',image\_inner)

key = cv2.waitKey(1) # 刷新率

if key == 32:

break # 空格键退出while循环

if retval\_outer :

cv2.imshow('outer',image\_outer)

key = cv2.waitKey(1) # 刷新率

if key == 32:

break # 空格键退出while循环

capture\_inner.release()

capture\_outer.release()

#### 是否打开isOpened

类的内置方法，打开摄像头成功返回True

ret = Capture.isOpened()

#### 读取帧read

Isread,image = Capture.read()

读取帧是否成功和帧

读取速率取决于cv2.waitKey()的刷新速度

#### 关闭release

Capture.release()

#### 获取视频属性get

Capture.get(propId)

帧数 / 秒数 = fps

proId的取值：

cv2.CAP\_PROP\_POS\_MSEC # 视频播放的位置 单位ms

cv2.CAP\_PROP\_POS\_FRAMES # 帧的索引从0开始

cv2.CAP\_PROP\_POS\_AVI\_RATIO #视频文件的相对位置 0表示开始播放 1结束

cv2.CAP\_PROP\_FRAME\_WIDTH # 帧宽度

cv2.CAP\_PROP\_FRAME\_HEIGHT # 帧高度

cv2.CAP\_PROP\_FPS # 帧速率

cv2.CAP\_PROP\_FRAME\_COUNT # 帧数

cv2.CAP\_PROP\_FOURCC # 4个字符表示的视频编码格式

cv2.CAP\_PROP\_FORMAT # retrieve方法返回的Mat对象的格式

cv2.CAP\_PROP\_MODE # 当前捕获模式的后端专用值

cv2.CAP\_PROP\_CONVERT\_RGB # 是否图像转为RGB

### 保存视频VideoWriter

Output=VideoWriter([filename, fourcc, fps, frameSize[, isColor]]) -> <VideoWriter object>

Filename：保存的视频路径

Fourcc：4个字符表示的视频编码格式

Fps：帧率

FrameSize：每帧的大小

#### 保存write

Output.write(frame)

frame是摄像头或者视频捕捉到的图像帧，即Capture.read()

#### 关闭release

Output.release()

### 视频编码格式VideoWriter\_fourcc

cv2.VideoWriter\_fourcc('I','4','2','0') 格式见P221页

### 读取分类器文件CascadeClassifier

分类器文件xml包含正面人脸、正面猫脸、侧面人脸、人下半身和上半身、人左眼和右眼、笑容、车牌、眼睛、身形和眼睛等xml

CascadeClassifier([filename]) -> <CascadeClassifier object>

返回指定文件的分类器对象CascadeClassifier

#### 检测输入图像detectMultiScale

Cascade = cv2.CascadeClassifier(filename)

Cascade.detectMultiScale()

detectMultiScale(image[,scaleFactor[,minNeighbors[,flags[,minSize[,maxSize]]]]]) -> objects

scaleFactor：扫描的图像的缩放比例

minNeighbors：每个候选区域至少保留多少个检测结果才能判定为人脸，越大误差越小

minSize：最小的目标尺寸

maxSize：最大的目标尺寸

Flags：旧版本的参数，使用默认值

返回目标区域的数组，每个元素包含4个值，表示左上角坐标和区域的宽高

### 绘制关键点drawKeypoints

## CV2(C++)

这里一律使用C++风格的函数，不包含C风格，即不是驼峰命名的cvWaitkey，而是使用waitkey

### 安装C++平台

#### 官网已经编译好的包

官网下载安装包后解压到目录：<https://opencv.org/releases/>

① 电脑-属性-高级-添加环境变量-添加用户变量和系统变量

D:\opencv4.3.0\opencv\build\bin

D:\opencv4.3.0\opencv\build\x64\vc15\bin

② 新建1个项目，注意用英文目录

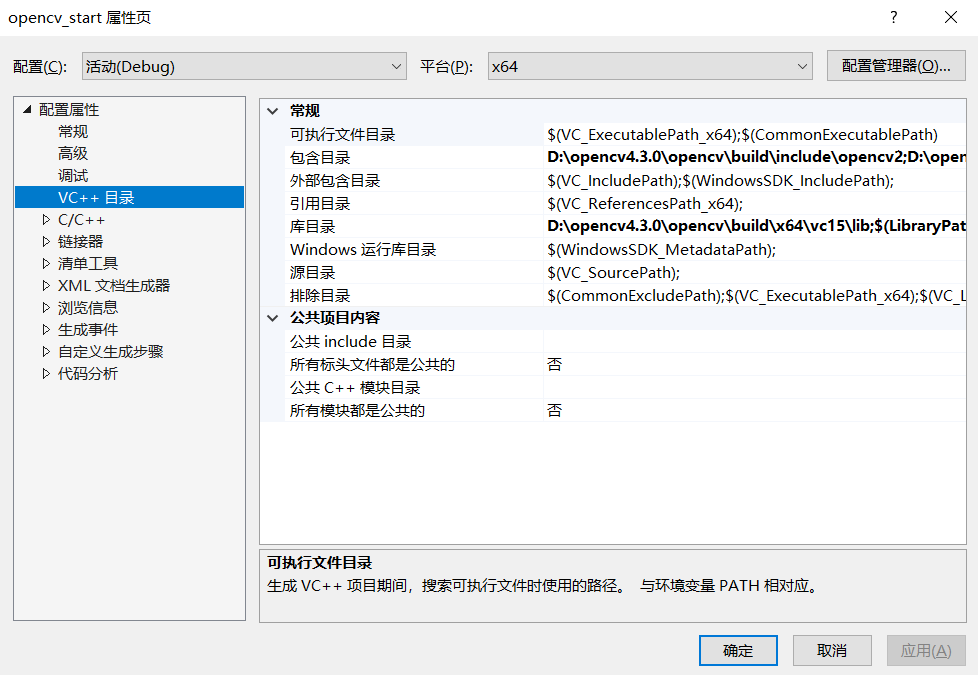
右击项目属性弹性界面，找到左边的VC++目录，注意debug平台必须选为x64

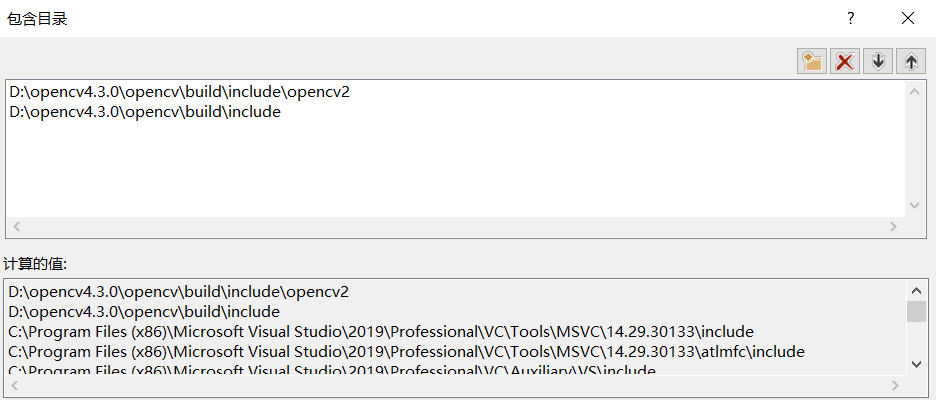
在包含目录中添加opencv的以下解压文件路径

D:\opencv4.3.0\opencv\build\include

<D:\opencv4.3.0\opencv\build\include\opencv>不用

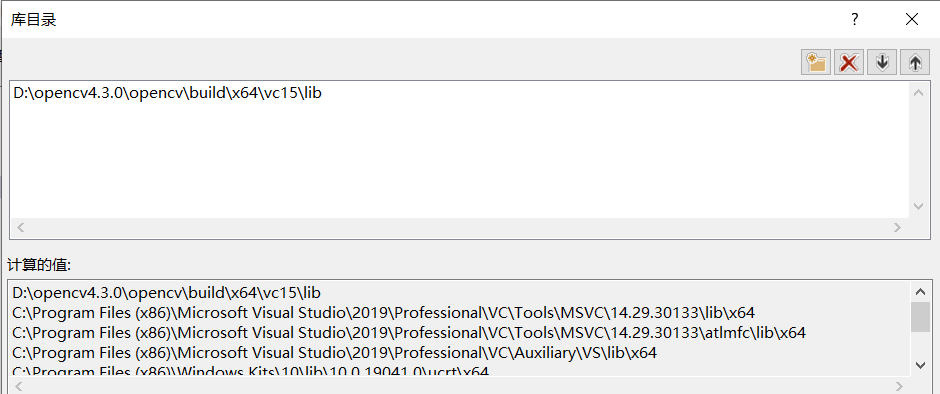
D:\opencv4.3.0\opencv\build\include\opencv2



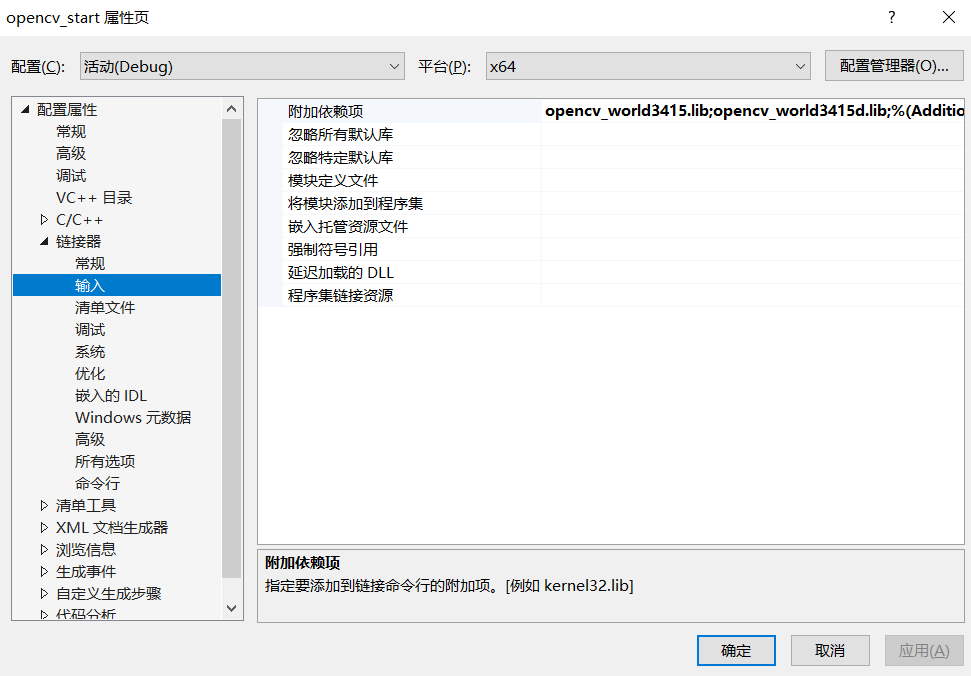


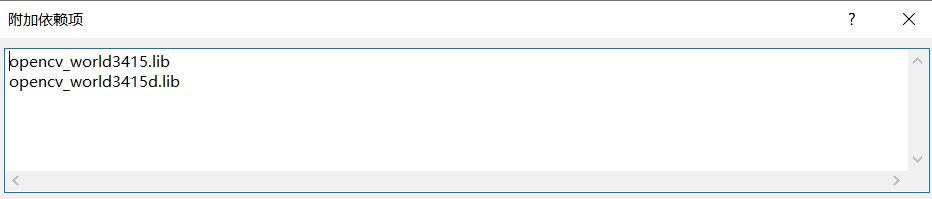
在库目录中添加以下路径，vc15和vc14选1个即可

D:\opencv4.3.0\opencv\build\x64\vc15\lib



然后找到链接器的输入窗口，找到附加依赖项





注意：要添加的是库目录路径下vc15的这2个lib，不要把vc14的lib添加进来

opencv\_world3415.lib

opencv\_world3415d.lib

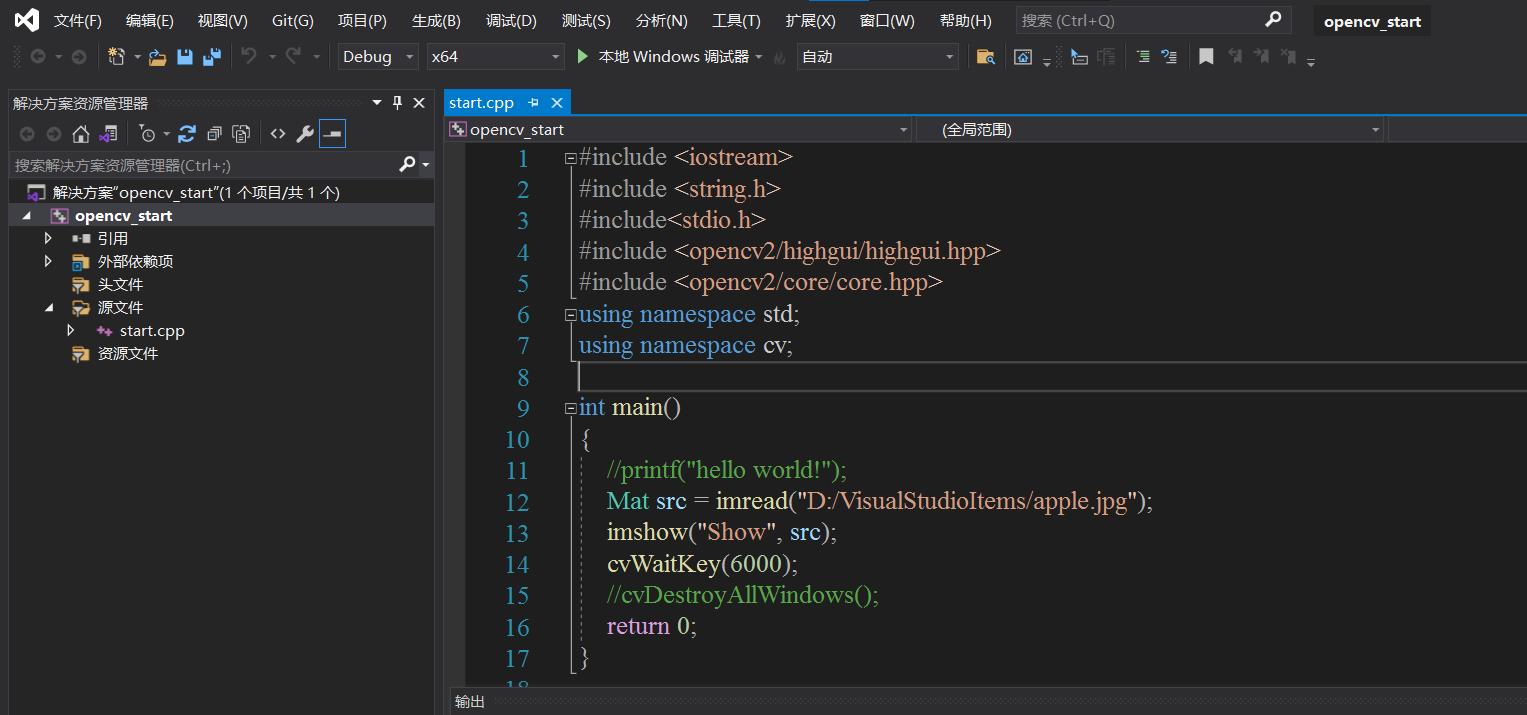
③ 把自己opencv文件目录下的\opencv\build\x64\vc15\bin中的三个dll文件，拷贝到C:\Windows\System32 中，重启电脑

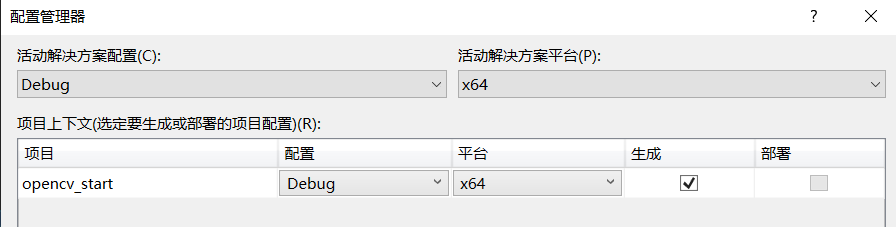
④ 测试：注意debug必须改为x64才正确

#include <opencv2/highgui/highgui.hpp> //不要带空格

#include <opencv2/core/core.hpp>

using namespace cv;





#### opencv和contrib共同编译(4.5.4版本)

接下来说明opencv4.5.4和opencv\_contrib-4.5.4的下载和编译

下载地址：<https://github.com/opencv>

cmake下载地址：<https://cmake.org/download/>

①下载版本相同的文件，解压

②创建一个文件夹opencv4，把opencv4.5.4和opencv\_contrib-4.5.4两个文件夹都放在这里

③接下来进入opencv4.5.4文件夹，创建source和build文件夹，把那些代码进一步剪贴到source文件夹，source有CMakeLists.txt文件

文件开头输入set (WITH\_GTK 0)

④打开cmake，选择vs2019编译

⑤勾选BUILD\_opencv\_world

⑥勾选OPENCV\_ENABEL\_NONFREE

⑦找到OPENCV\_EXTRA\_MODULES\_PATH，

输入D:/opencv4/opencv\_contrib-4.5.4/modules，然后继续点击configure，之后generate即可

⑧generate done后打开open project，直接右击生成解决方案等待即可

⑨然后在CMakeTargets文件夹找到INSTALL这项，右键->仅用于此项目->仅生成INSTALL，这样就在build目录下生成了一个名为install的目录，以后工程中需要配置的文件都在里面

⑩配置环境变量

D:\opencv4\opencv4.5.4\build\install\x64\vc16\bin

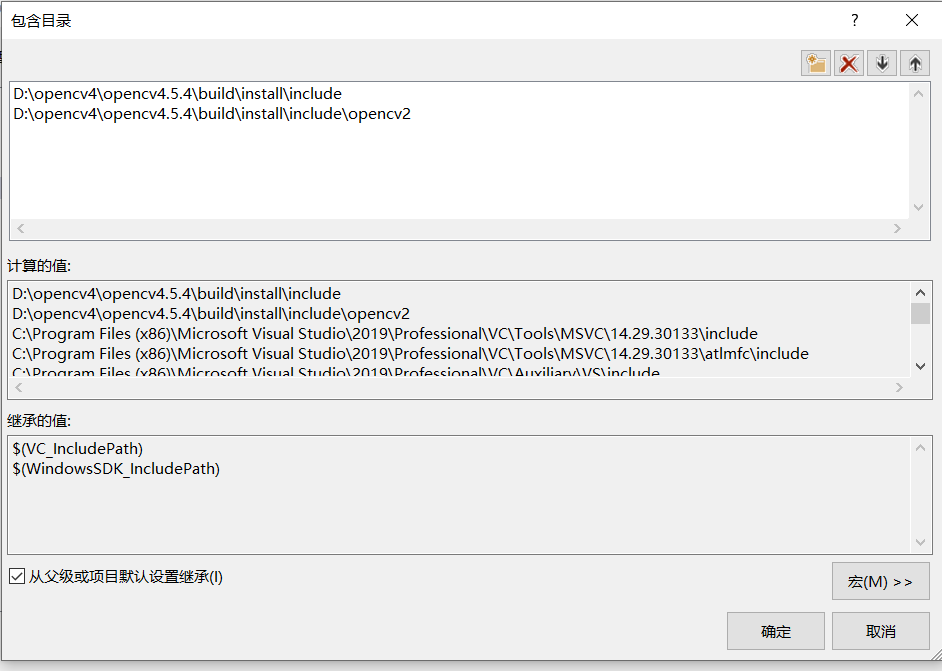
D:\opencv4\opencv4.5.4\build\install\x64\vc16\lib

十一、

D:\opencv4\opencv4.5.4\build\install\include\opencv2

D:\opencv4\opencv4.5.4\build\install\include

都加入包含目录



库目录：

D:\opencv4\opencv4.5.4\build\install\x64\vc16\lib

链接器-输入->找到库目录的两个lib文件

opencv\_img\_hash454d.lib

opencv\_world454d.lib

#### opencv和contrib共同编译(3.4.0版本)

①在cmake的OpenCVDetectCXXCompiler.cmake文件中插入这段话(147行寻找)，解决链接：<https://blog.csdn.net/qq_34851605/article/details/104432994>

elseif(MSVC\_VERSION EQUAL 1929)

set(OpenCV\_RUNTIME vc16)

elseif(MSVC\_VERSION MATCHES "^193[0-9]$")

set(OpenCV\_RUNTIME vc17)

②cmakelist文件中输入：set (WITH\_GTK 0)

仿照设置几个参数，注意额外的路径是contrib的modules

D:\opencv3.4.0\opencv\_contrib-3.4.0\modules

③使用wifi下载，不要使用数据移动

出现网络问题可能的解决方案：

<https://blog.csdn.net/KayChanGEEK/article/details/79919417>

<https://raw.githubusercontent.com/opencv/opencv_3rdparty/66b1fed06cf3510235f367f96aa26da5cb234a15/ffmpeg/opencv_ffmpeg.dll>

<https://raw.githubusercontent.com/opencv/opencv_3rdparty/66b1fed06cf3510235f367f96aa26da5cb234a15/ffmpeg/opencv_ffmpeg_64.dll>

<https://raw.githubusercontent.com/opencv/opencv_3rdparty/66b1fed06cf3510235f367f96aa26da5cb234a15/ffmpeg/ffmpeg_version.cmake>

从上述3个地址下载对应的文件，最后要改名，放在地址

D:\opencv3\opencv-3.4.0\source\.cache\ffmpeg

分别改名为：

3ae76b105113d944984b2351c61e21c6-opencv\_ffmpeg.dll

cf3bb5bc9d393b022ea7a42eb63e794d-opencv\_ffmpeg\_64.dll

ec59008da403fb18ab3c1ed66aed583b-ffmpeg\_version.cmake

cmake文件记事本打开，输入：

set(FFMPEG\_libavcodec\_FOUND 1)

set(FFMPEG\_libavformat\_FOUND 1)

set(FFMPEG\_libavutil\_FOUND 1)

set(FFMPEG\_libswscale\_FOUND 1)

set(FFMPEG\_libavresample\_FOUND 1)

set(FFMPEG\_libavcodec\_VERSION 57.89.100)

set(FFMPEG\_libavformat\_VERSION 57.71.100)

set(FFMPEG\_libavutil\_VERSION 55.58.100)

set(FFMPEG\_libswscale\_VERSION 4.6.100)

set(FFMPEG\_libavresample\_VERSION 3.5.0)

0421e642bc7ad741a2236d3ec4190bdd-ippicv\_2017u3\_win\_intel64\_general\_20170822.zip

④Windows下OpenCV，加载\*\_highgui\_gtk453\_64.dll失败，问题原因？

<https://www.zhihu.com/question/482457348>

⑤其他和2.4.1.2过程类似

### 基本的数据类型

#### 概述

cv::Vec<>，固定向量类，向量可以承载2,3,4,6个元素，类型有b(unsigned char)、w(unsigned short)、s(short)、i(int)、f(float)、d(double)

所以固定向量类有很多的别名，cv::Vec{2,3,4,6}{b,w,s,i,f,d}，例如cv::Vec2i表示具有2个整型元素的固定向量，访问元素通过下标访问myVec[0]、myVec[1]

cv::Point<>，固定点类，只有2-3个元素，用来描述二维、三维坐标，与固定向量类不同在于通过句号运算来访问，即myPoint.x、myPoint.y，常用cv::Point2f

cv::Scalar<>，在固定点类基础上实现，本质上是四维Point类，一般使用双精度，继承于cv::Vec<double,4>，但是是通过下标访问的

cv::Size<>，和固定点类很像，区别是属性为width、height而不是x、y，因为高度宽度是整数，实际上使用的是cv::Size2i，浮点类型也可以cv::Size2f

cv::Rect<>，是处理整数类型的矩形，有坐标和宽高

cv::RotateRect<>，用于处理非轴对称的矩形，有一个cv::Point2f类型的中心点，一个cv::Size2f的size，还有一个额外的浮点类型的角度

#### cv::Point类

### cv::Mat类

#### 类成员函数

##### ones

常用的函数重载

int (rows, int cols, int type)

ones(Size size, int type)

### 函数索引

#### waitKey

waitKey(4000);

或者

waitKey(-1);

system("pause");

#### nameWindow

定义一个带名字的窗口

相关的参数常量宏定义为：WINDOW\_，如WINDOW\_FREERATIO

namedWindow("input", WINDOW\_FREERATIO);//可调的自由比例

#### imread

相关的参数常量宏定义为：IMREAD\_，如IMREAD\_GRAYSCALE

读取的图像具有depth

# 图像处理应用

## 图像处理

### Opencv的坐标系

坐标系向下Y，向右X，对应的是数组的行索引和列索引

如行为5，列为9的数组索引表示在（10，6）的像素

数组行索引 = 像素所在行数 - 1 = 像素纵坐标

数组列索引 = 像素所在列数 - 1 = 像素横坐标

Opencv的三原色符合光学三原色，即红绿蓝=白色，颜料中的三原色为红黄蓝=黑色

### HSV空间

色调指光的颜色，opencv中取值范围[0,180]，红黄绿蓝依次对应0、30、60、90；

饱和度为色彩的深浅，在[0,255]选择，表示色彩的深浅，饱和度为0即是灰度图像

亮度为光的明暗，亮度越大越亮，为0时趋于纯黑色

### 滤波器分类

均值滤波：低通滤波

中值滤波：低通滤波，适用于椒盐噪声、脉冲噪声

高斯滤波：低通滤波，高斯噪声

双边滤波：低通滤波，保护边界信息

### 高斯核的size和sigma

sigma变大以后，峰值变小，分布更分散，也就是说图片中心像素会被周边更多的像素影响，从而平滑的更厉害

如果sigma一样，峰值是相同的，此时size变大，受中心的影响也越小，分布广泛所以平滑能力强，如果size变小受中心影响越强，那么也就越接近原图像，平滑能力差

经验法则：卷积核的半窗宽度设置为3sigma，模板尺寸为6sigma+1

小sigma连续多次的计算复杂度与直接使用大sigma的复杂度效果相同且运算量低，例如sigma=1，那么size取7，一次滑窗计算49次乘法

## 图像分类

## 图像定位

## 目标检测

一个图片有1个目标可以先被分类，分类以后还要返回边界框，可以看成是回归问题，如果有多个目标可以认为是目标检测问题

返回的输出向量一般为8个条目组成，首先是否是检测目标，其次是目标的类别（使用二进制，例如一张图片有小车、卡车、行人等），最后还要返回目标的位置，使用bx、by、bw、bh替代，即到坐标系的距离以及边界框的高度和宽度。

### 图像匹配

#### 基于随机抽样一致性的哈里斯角点匹配

#### 基于BRIEF

#### 基于SIFT

#### 基于ORB

#### 基于类Haar特征的人脸匹配

## 图像分割

### 语义分割

像素级的分类，每个像素都需要分类，识别出属于的类别

### 实例分割

那么实例分割不但要进行像素级别的分类，还需在具体的类别基础上区别开不同的实例

### 应用

#### 基于霍夫变换

#### 基于二值化和otsu

#### 基于边缘

#### 基于形态学分水岭

#### 基于菲尔森茨瓦布高效图

#### 基于k均值聚类slic

#### 基于快速移位quickshift

#### 基于紧凑型分水岭

#### 基于区域生长

#### 基于活动轮廓

#### 基于形态学蛇

#### 基于图论GrabCut

#### 基于谱聚类SpectralClustering

sklearn.cluster.spectral\_clustering()

用于将图像分为前景和背景，一可以使用聚类中心为2的K均值比对

### 图像增强和恢复

### 动作识别

### 姿态估计

## 语义分割

# 机器学习模型

## 经典模型

首先介绍无监督学习PCA、K均值聚类

其次说明有监督学习

使用sklearn实现

### PCA

### K均值聚类

### 回归

#### 一元线性回归

一元线性回归y=ax+b

a可以通过最小二乘法公式得到，公式如下

b的公式为

可以使用R2来衡量误差，公式如下

其实分子就是绝对误差平方累计和，也是均方差损失mse；分母为偏离均值程度的平方和，其实就是方差var

#### 多元线性回归

y\_pred拟合值与输入值满足的关系如下：

y\_pred = a0 + a1x1 + a2x2 + ... + anxn ①

将方程①写为如下形式，其中x0为全1列向量

y\_pred = a0x0 + a1x1 + a2x2 + ... + anxn ②

x1,x2,x3,...xn都是输入向量 ,例如波士顿房价为 506 \* 13 的矩阵

[ 1 x1(1) x2(1) ... xn(1)]

X = [ 1 x1(2) x2(2) ... xn(2)]

[ ... ... ]

[ 1 x1(n) x2(n) ... xn(n)]

a = [a0,a1,..an]^T a是拟合系数列向量

对于方程②如果希望找到1个最佳拟合的方程，其系数应当满足

其中y\_ture为与x1,x2,...,xn对应的实际值

a = (X^T\*X)^(-1) \* X^T \* y\_true = a\_ \* a\_\_ \* a\_\_\_ ③

设输入的形状为m\*n，((n+1,m)\*(m,n+1))^-1 --->(m,m) \* (m,n+1)--->(m,n+1) \* (m,)-->(n+1,)

最后一步是m×n的矩阵的每列和(m,)的列向量分别点积，得到(n+1,)的向量 对应n+1个拟合系数[a0,a1,...an]

其中a0就是截距系数，剩下的都是斜率系数a1-->an

#### 逻辑回归

逻辑回归应用于二分类问题，即y={0,1}而非零散的取值，输出y\_pred再经过sigmoid转换为y\_predict={0,1}

逻辑预测值y\_predict依据概率p来输出0或1

那么损失函数实际上希望真实值为0时，p越大(预测接近1)损失越大；真实值为1时，p越小(预测越接近0)损失越大，所以对于单个样本损失函数可以写为

其中y是真实值，取值为0或1

而p实际上就是对拟合方程的预测输出使用sigmod进行归一化，即

y\_predict实际上相当于把p概率向量转为非0即1的整型向量

最终对于m个样本损失函数的形式为

### 树回归

### 决策树

### 朴素贝叶斯

### 随机森林和AdaBoost

### 支持向量机

### KNN近邻

核心思想是，如果一个样本在特征空间中的K个最相邻的样本中的大多数属于某一个类别，则该样本也属于这个类别，并具有这个类别上样本的特性

涉及到的超参数有L和K，最好的L、K依赖于具体问题

如果同一张图片做了一些颜色改变，遮挡或者裁剪、仿射等，KNN不会将其归类为同一组标签，也就是不适用于图片视觉上的偏差，或者说一张图片的不同形状可能会有相同的距离

同时KNN对样本的分布没有任何预设，所以能够准确应用的前提是样本足够密集，但是在高维空间中需要的样本数指数增长，才能足够密集，这样计算成本是非常大的

步骤：

①准备数据，对数据进行预处理

②计算测试样本点（也就是待分类点）到其他每个训练样本点的距离

计算每个测试样本和所有训练样本的距离，当训练样本有N个时1个测试样本得到N个距离L

③对每个距离进行排序，然后选择出距离最小的K个点

选取一个测试样本N个距离中的K个，这K个对应的训练样本的类别的大多数即认为是该测试样本的类别

④对K个点所属的类别进行比较，根据少数服从多数的原则，将测试样本点归入在K个点中占比最高的那一类

一个测试样本完成后，继续重复直到测试样本全部完成，所以实际上有2个循环

# k个最近邻

# x\_train.shape[0] = N

#伪代码:

开始

对测试样本的每个样本循环

计算该样本和所有训练样本的距离，长度为N（广播效应）

对当前样本的N个距离进行升序，排序之前训练样本的默认位置0,1,..N

排序的同时也要记录排序顺序

选择距离的前k个，作为最小的距离

根据排序顺序得到这k个距离对应的训练样本所对应的标签

开始投票

记录每个标签出现的次数

生成投票向量，将最大值索引确定的标签认为是当前样本的标签

for i in range(x\_test.shape[0]): # 每个测试样本

# 利用广播效应，不要使用for j in range(x\_train.shape[0])

dist = np.sum(np.abs(x\_train - i)) # L1距离,dist存放了N个距离,对应了训练样本的每行0,1,2...N

dist = pd.Series(dist) # 利用排序可以记录index的功能

sorted\_dist = dist.sort\_values(ascending=True) # 默认升序

k\_sorted\_dist = sorted\_dist.index.array [:k]# 找到前k个对应的位置

Label = [ [] for \_ in range(len(set(y\_train)))]

for j in k\_sorted\_dist : # 对这k个位置寻找对应的标签

label = y\_train(j) # 例如label =[0,1,2,3,...9]

/# 如果标签为1则 1的位置+1

Label[label] += 1

idx\_max = np.argmax(Label) # 投票数最多的标签

print(“当前样本的预测结果为：”,idx\_max)

## Sklearn库

### utils模块

#### 随机打乱数组shuffle

shuffle(im,random\_state=0,n\_samples=1000)

n\_samples表示抽取n个出来，可以输入二维数组

### linear\_model模块

#### 线性回归LinearRegression

lr = LinearRegression(fit\_intercept=True,normalize=False,copy\_X=True)

fit\_intercept：是否计算此模型的截距。如果设置若为False，计算中将不使用截距（即数据应居中）。

normalize：当fit\_intercept设置为False时，被忽略。如果为True，则在回归之前，将通过减去平均值并除以L2范数。可以使用sklearn.preprocessing.StandardScaler进行标准化

copy\_X：如果为True，将复制X；否则，它可能会被覆盖

具有属性y=mx+b

m = lr.coef\_

b = lr.intercept\_

### cluster模块

#### K均值KMeans

返回聚类中心坐标和实际标签，分别表示聚类数据的实际标签和指定聚类的中心坐标。

kmeans = KMeans(n\_clusters=n\_clusters, random\_state=0).fit(train\_x)

x\_pred = kmeans.predict(x)

每个像素点预测的结果，属于何种标签，标签和聚类中心一一对应

可以遍历每个像素点的预测值得到点的标签：

current\_pixel\_label = x\_pred[idx]

再用这个标签得到对应的聚类中心：

current\_pixel = cluster\_centers[current\_pixel\_label]

对于图片数据，找到了聚类中心就可以将属于这类的像素点都替换为聚类中心像素值：

img\_re[i, j] = current\_pixel

cluster\_centers = kmeans.cluster\_centers\_ # 聚类中心坐标

cluster\_labels = kmeans.labels\_ # 聚类数据的实际标签

#### 小样本K均值MiniBatchKMeans

Mini Batch K-Means算法是K-Means算法的变种，采用小批量的数据子集减小计算时间，同时仍试图优化目标函数，这里所谓的小批量是指每次训练算法时所随机抽取的数据子集，采用这些随机产生的子集进行训练算法，大大减小了计算时间，与其他算法相比，减少了k-均值的收敛时间，小批量k-均值产生的结果，一般只略差于标准算法

#### 谱聚类SpectralClustering

谱聚类是从图论中演化出来的算法，后来在聚类中得到了广泛的应用。它的主要思想是把所有的数据看做空间中的点，这些点之间可以用边连接起来。距离较远的两个点之间的边权重值较低，而距离较近的两个点之间的边权重值较高，通过对所有数据点组成的图进行切图，让切图后不同的子图间边权重和尽可能的低，而子图内的边权重和尽可能的高，从而达到聚类的目的

比起传统的K-Means算法，谱聚类对数据分布的适应性更强，聚类效果也很优秀，同时聚类的计算量也小很多，常用于图像分割

### neighbors模块

#### 球树最近邻BallTree

#### KD树最近邻KDTree

#### K近邻KNeighborsClassifier

KNeighborsClassifier(n\_neighbors=5,weights='uniform',algorithm='auto',leaf\_size=30,p=2, metric='minkowski', metric\_params=None, n\_jobs=None)

### naive\_bayes模块

#### 高斯朴素贝叶斯GaussianNB

gb = GaussianNB(priors=None,var\_smoothing=1e-9)

priors：类的先验概率。如果指定，则不指定优先级根据数据进行调整。

如果不指定，某类的先验概率的计算方式为该类的总样本数/总样本数,给出的话以priors为准

var\_smoothing：为计算稳定性而添加到方差中的所有特征的最大方差部分。

属性值：

class\_count\_：每个类别观察到的训练样本数量，shape (n\_classes,)

class\_prior\_：每个类别发生的先验概率，shape (n\_classes,)

classes\_：分类器已知的类标签，shape (n\_classes,)

epsilon\_：float，方差数值稳定性

sigma\_：每类特征的方差，shape (n\_classes, n\_features)

theta：每类每个特征的平均值，shape (n\_classes, n\_features)

fit函数：X,y的shape分别是(n\_samples, n\_features)和(n\_samples,)

预测有三种方法，包括predict，predict\_log\_proba和predict\_proba

GaussianNB一个重要的功能是有 partial\_fit方法，这个方法的一般用在如果训练集数据量非常大，一次不能全部载入内存的时候。这时我们可以把训练集分成若干等分，重复调用partial\_fit来一步步的学习训练集

#### 多项式朴素贝叶斯MultinomialNB

#### 伯努利朴素贝叶斯)BernoulliNB

### svm模块

#### 支持向量机SVC

svm=SVC(C=1.0,kernel='rbf',degree=3,gamma='scale',class\_weight=None,random\_state=10)

C：正则化系数

kernel：{'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'}

degree：只用于'poly'

gamma：高斯核系数，适用于rbf

class\_weight：是字典，形如{0:1,1:10}，类别1的样本数是类别0的10倍，所以惩罚系数也越大，还可以使用字符串"balanced"自动计算不平衡样本

model = svm.SVC(kernel="rbf",

class\_weight={0:1,1:n\_multiple},random\_state=0)

### ensemble模块

#### 集成分类器AdaBoostClassifier

AdaBoostClassifier(base\_estimator=None, \*,

n\_estimators=50,

learning\_rate=1.,

algorithm='SAMME.R',

random\_state=None)

base\_estimator：从中生成集合的基估计量

n\_estimators：拟合子估计量的集合数量

learning\_rate：学习率通过以下方式缩小每个分类器的贡献，学习率”和

``n\_估计量```

algorithm：{'SAMME', 'SAMME.R'}，如果选用第2中base\_estimator必须指定类概率的计算。SAMME.R算法通常比SAMME收敛得更快，更少的boosting迭代实现更低的测试错误。

例子：

X, y = make\_classification(n\_samples=1000, n\_features=4,

... n\_informative=2, n\_redundant=0,

... random\_state=0, shuffle=False)

clf = AdaBoostClassifier(n\_estimators=100, random\_state=0)

clf.fit(X, y)

clf.score(X, y)

### manifold模块

#### t-分布式随机邻域嵌入TSNE

相比PCA，TSNE 运行极慢，属于高维数据的非线性降维算法，前者是线性的。TSNE保留的属性信息，更具代表性，也即最能体现样本间的差异

TSNE(n\_components=2)

tsne= sklearn\_manifold.TSNE(n\_components=2).fit\_transform(d3.data.numpy())

pca=sklearn\_decompose.PCA(n\_components=2).fit\_transform(d3.data.numpy())

### decomposition模块

#### 主成分降维PCA

pca = PCA（n\_components）

属性：

components\_：输入数据格式为样本数\*输入维数，输出格式为样本数\*n\_components，components\_为转换矩阵的转置，格式为n\_components\*输入维数，也就是前components\_个主要的成分，它们相互正交，在图像降维中可以认为是特征脸。

降维后矩阵=降维前矩阵\*components\_^T，即input = output \* components，而output可以看成权重，output的每行对应每个样本，每列是每个样本在特征脸上的权重。

方法：

x\_pca = pca.fit\_transform(x)

x\_re\_0 = np.reshape(x\_pca[0]@pca\_components\_ ,(64,64))

inverse\_transform的实现机制就是矩阵乘法，每个输出样本作为权重乘以转换矩阵就是对应样本的重建图像

### metrics模块

#### 分类报告classification\_report

classification\_report(y\_true, y\_pred)

#### 解释方差回归得分explained\_variance\_score

explained\_variance\_score(y\_true,y\_pred)

#### 准确率得分accuracy\_score

accuracy\_score(y\_true, y\_pred)

train\_x,test\_x,train\_y,test\_y=train\_test\_split(X,y,train\_size=0.7,test\_size=None, random\_state=10,shuffle=True,stratify=None)

stratify适用于样本不平衡的数据集，=y可以使划分数据集前后的比例不变，划分前两类是4比1，分割后依然4比1

#### R2得分r2\_score

用于回归，得分范围在0-1，1是最好的，表明的自变量完美的预测了因变量，为0说明不能进行预测该值

r2\_score(y\_true,y\_pred,\*,sample\_weight=None,multioutput="uniform\_average")

#### 均方误差mean\_squared\_error

用于回归

mean\_squared\_error(y\_true,y\_pred,\*,sample\_weight=None,multioutput="uniform\_average",squared=True)

#### 混淆矩阵confusion\_matrix

需要y\_true和y\_pred参数，常见的用法如下

cm = confusion\_matrix(y\_test,LR.predict(X\_test))

cm = pd.DataFrame(cm)

sns.set\_style("whitegrid") # 默认白色背景

sns.set(font\_scale=1.2) # 字体

sns.heatmap(cm,annot=True,annot\_kws={"size":16},fmt="d",cbar=True) # 单元格写入数值和绘制颜色条

### pipeline模块

#### 通道Pipline

Pipeline(steps,memory=None)

参数：

Steps：按链接顺序链接的（名称，变换）元组（实现拟合/变换）列表，最后一个对象是估计器

Memory：默认情况下，不执行缓存。 如果给出字符串，则它是缓存目录的路径

属性：

named\_steps：可以找到每个步骤执行后的一些属性

方法：

fit(X,y)

transform(X,y)

fit\_transform(X,y)

fit\_predict(X,y)

predict(X)

fit(): Method calculates the parameters μ and σ and saves them as internal objects.

解释：简单来说，就是求得训练集X的均值，方差，最大值，最小值,这些训练集X固有的属性。

transform()：在fit的基础上，进行标准化，降维，归一化等操作（看具体用的是哪个工具，如PCA，StandardScaler等）

fit\_transform()：fit\_transform是fit和transform的组合，既包括了训练又包含了转换

transform()和fit\_transform()二者的功能都是对数据进行某种统一处理（比如标准化~N(0,1)，将数据缩放(映射)到某个固定区间，归一化，正则化等）

例子：

poly\_features = PolynomialFeatures(degree,include\_bias=False)

linear\_regression = LinearRegression()

pipline = Pipeline([("poly",poly\_features),("lr",linear\_regression)])#管道串联模型，要求输入是列表且列表元素是元组，分别是自定义的步骤名称和串联的模型实例

### datasets模块

#### 手写数字digits

digits = load\_digits()

X = data["data"] # 已经展平的数据

y = data["target"] #0-7共8类

#### 加利福尼亚房价house

house = fetch\_california\_housing()

#### 鸢尾花load\_iris

iris = load\_iris()

#### 波士顿房价boston

data = load\_boston()

X ,y = load\_boston(return\_X\_y=True)

#### 月亮数据集make\_moons

data=make\_moons(n\_samples=n\_samples,shuffle=True,noise=0.5,random\_state=10)

X ,y = data[0],data[1]

#### 分类数据集make\_classification

make\_classification(n\_samples=100, n\_features=20, \*, n\_informative=2,

n\_redundant=2, n\_repeated=0, n\_classes=2,n\_clusters\_per\_class=2, weights=None, flip\_y=0.01,class\_sep=1.0, hypercube=True, shift=0.0, scale=1.0,shuffle=True, random\_state=None)

n\_samples：数据样本数，default=100

n\_features：特征维数，default=20

n\_informative：信息特征的数量

n\_redundant：冗余功能的数量。这些特征生成信息特征的随机线性组合。

n\_repeated：从信息模型中随机抽取的重复特征数以及冗余功能。

n\_classes：分类问题的类（或标签）数。

n\_clusters\_per\_class：每个类的群集数

weights：shape (n\_classes,) or (n\_classes - 1,)，分配给每个类别的样本比例，如果len（weights）==n\_classes-1，则自动推定最后一个权重

flip\_y：随机分配类别的样本分数。更大的值在标签中引入噪波并进行分类

任务更艰巨，注意，在某些情况下，设置flip\_y>0可能导致y中的类少于n个

class\_sep：乘以超立方体大小的因子，越大越容易分类

hypercube：True：簇被放置在超立方体的顶点上；Fasle，簇放置在随机多面体的顶点上

shift：按指定值移动特征如果没有，则显示特征被[-class\_sep，class\_sep]中绘制的随机值移位

scale：将要素乘以指定的值。如果没有，则通过[1100]中绘制的随机值缩放特征。请注意，平移后会发生缩放。

shuffle： Shuffle the samples and the features

### preprocessing模块

#### 标准差标准化StandardScaler

处理方法：标准化数据减去均值，然后除以标准差，经过处理后数据符合标准正态分布，即均值为0，标准差为1，适用于本身服从正态分布的数据

x = (x-mean) / std

ss = sklearprocess.StandardScaler(copy=True, with\_mean=True, with\_std=True)

内置方法：

fit\_transform(X=X, y=None)

#### 极差标准化 MinMaxScaler

适用于分布范围较稳定的数据，当新数据的加入导致max/min变化，则需重新定义，处理方法：将特征缩放到给定的最小值和最大值之间，也可以将每个特征的最大绝对值转换至单位大小。这种方法是对原始数据的线性变换，将数据归一到[0,1]中间，x = (x-min) / (max-min)

MinMaxScaler(feature\_range=(0, 1), copy=True)

### model\_selection模块

#### 划分训练测试集train\_test\_spilit

train\_test\_spilit(data,target,train\_size=0.7,test\_size=None,random\_state=10,shuffle=True,stratify=None)

返回train\_x,test\_x,train\_y,test\_y

#### K折交叉验证KFold

kfold = KFold(n\_splits=10,random\_state=10,shuffle=True)

#### 交叉验证得分cross\_val\_score

cross\_val\_score(estimator, X, y=None, \*, groups=None, scoring=None,cv=None, n\_jobs=None,verbose=0,fit\_params=None,pre\_dispatch='2\*n\_jobs',error\_score=np.nan)

需要指定估计器对象、交叉验证的数据和标签，cv为交叉验证对象，返回K个交叉验证分数

scores = cross\_val\_score(estimator=knn, X=digits.data, y=digits.target,cv=kfold)

#### 交叉验证cross\_validate

cv\_results=cross\_validate(estimator,X,y=None,groups=None,scoring=None,cv=None,n\_jobs=None,verbose=0,fit\_params=None,pre\_dispatch='2\*n\_jobs',return\_train\_score=False,return\_estimator=False, error\_score=np.nan)

score：指定评估的参数，不指定则使用估计器的score方法，也就是acu，可以指定('r2', 'neg\_mean\_squared\_error')等

return\_train\_score：是否返回训练分数

fit\_params：参数传递到估计器的拟合方法

return\_estimator：是否返回每次的估计器

返回：默认['fit\_time', 'score\_time', 'test\_score']，根据score指定的会返回对应的得分，return\_train\_score也会返回train\_score

#### 网格搜索GridSearchCV

estimator：指定选用的估计器

param\_grid：配套的估计器网格参数：字典，关键字是参数名，值是列表

cv：交叉验证次数，默认5

iid ：为真时返回交叉验证的平均测试集得分,默认False

scoring ：评价测试集的指标，默认是准确率

<https://blog.csdn.net/qq_41076797/article/details/102755893>查看相关可选指标

return\_train\_score：计算训练的得分，默认False，此时属性cv\_results\_不会返回训练得分

refit：默认为True，会在整个数据集上使用最佳参数重新调整估计器

GridSearchCV(estimator=SVC(),

param\_grid={'C': [1, 10], 'kernel': ('linear', 'rbf')})

# 深度学习模型

深度学习无需提取图像特征，直接将图像像素作为特征，最简单情况是将其拉伸为1维

图像和某个卷积核的卷积就是一种特征，如果多个卷积核得到的是多种特征，例如构建了卷积核组，可以检测多个方向的梯度，斑点等，但是是将这48个卷积核可以平均为多个卷积层，每层有多个卷积核，穿插BN层批归一化，或者非线性层RELU，以及池化层Pool也就是采样，最后得到了一个

对图像的一个像素而言可以得到多个特征，例如48个卷积核就能得到48维的特征向量，这个向量的最大值的位置代表了这个像素属于哪种特征

而且这个特征向量是一个稀疏向量，一个像素最多可能是几种可能的特征，所以可以使用压缩技术压缩特征

## 数据处理

### 预处理

去均值（保证和量纲无关），归一化

去相关（可以达到降维的目的，如果2个特征相关，其实用1个特征描述即可，求数据协方差矩阵），白化（将协方差矩阵变为单位矩阵）

### 划分数据集

训练集用来对给定的超参数进行训练得到模型，验证机用于对多组超参数进行验证，选择最好的超参数，测试集则是评估该模型的泛化能力

如果验证集的数量比较少，不能代表该组超参数的实际情况，可以使用K折交叉验证

例如3折交叉验证，抛去测试集后，将数据集分为了3份，对某组超参数进行3次验证，每份轮流作为验证集然后得分取平均，就得到这组超参数的平均得分，其他超参数对也是如此操作

## 损失函数类型Loss

首先以二分类问题说明损失函数，二分类问题真实值y\_true={0,1}或者{-1,1}，这里以y\_true={-1,1}进行说明

对于某个样本x，预测输出值y\_pre=w\*x，结果有正有负，经过sigmoid函数可以归一化到[0,1]之间，相应的y\_true根据-1或1的情况可以列表如下

预测大于0，且此时类别确实为1，那么预测正确，或者预测小于0，类别为-1，也是预测正确，其他情况均错误

|  |  |  |
| --- | --- | --- |
| y\_true\y\_pred | >0 | <0 |
| 1 | 1 | 0 |
| -1 | 0 | 1 |

根据y\_true\y\_pred的情况可以衍生出多种形式的损失函数，只需要y\_true\*y\_pred≥0即可，分别用y和p表示，例如8个样本

Eg : y = [1, -1, -1, -1, 1, -1, 1, 1] p=[0.4, 0.1, -0.5, -0.7, 0.6, -0.2, 0.1, 0.9]

### 0-1损失

损失非0即1，预测正确为0，否则为1

### 绝对值损失

绝对值损失函数是计算预测值与目标值的差的绝对值

### 对数损失

健壮性不强，相比于hinge loss对噪声更敏感，在逻辑回归中使用

对于单个样本，假设预测为M个类别的概率分别为*P*j(j=1,2,...M),实际类别为yj（一般情况下使用独热编码而非标签编码），那么对数损失为

如预测概率为[0.8,0.1,0.1],标签种类[1,0,0],该样本实际为1，那么损失为1\*log(0.8)+0\*log(0.1)+0\*log(0.1)

所有样本设有N个，那么其对数损失为

一般分类问题分为二分类和多分类，对于二分类问题，上述可以简化为

如预测概率为[0.8,0.2],标签为[1,0],那么损失为1\*log(0.8)+0\*log(0.2)

N分类问题可以拆分2个思路

第一种是构建N(N-1)/2个分类器，如[A,B,C]🡪AB,AC,BC三个分类器，然后每个分类器对样本进行预测，最后的投票结果服从多数，但是可能会出现某两个标签投票数相同，可以使用多次测试取平均结果避免

第二种是构建N个分类器，每次先假定一类标签为1，其他全部0，给出预测，如[A,B,C]🡪[A,BC],[AB,C],[AC,B]，同样可能多个分类器预测为1，那么需要引入置信度进一步判断，理想的结果是只有一个分类器预测为1，其他均为0

### 平方损失

常用于回归问题

### 指数损失

### Hinge损失

hinge损失函数表示如果被分类正确，损失为0，否则损失就为,SVM使用的该损失函数

### 多类SVM损失

单个样本损失=Σmax(0,错误分数*pij*-正确分数+1)

s为该样本的真实类别，*yis*为真实类别得分，*pij*为错误类别得分

第1个样本猫：max(0,-2.3-0.6+1)+max(0,1.9-0.6+1)=2.3，类似可计算其他样本，最后求和即可，注：正确类别得分比错误类别至少高1分以上损失才为0

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| y\_true\y\_pred | 猫 | 鸟 | 车 | 损失 |
| 猫 | 0.6 | -2.3 | 1.9 | 2.3 |
| 鸟 | 1.7 | 2.9 | 2.3 | 0.4 |
| 车 | 3.1 | -2.6 | 4.3 | 0 |

### 交叉熵损失

熵：H(p)=-Σp(x)log(p(x))

交叉熵：H(p,q)=-Σp(x)log(q(x))

相对熵：KL(p||q)=-Σp(x)log(q(x)/p(x))

相对熵也叫KL散度，用来度量两个分布之间的不相似性

H(p,q) = H(p) + KL(p||q)

如预测分布为q(x)=[0.1,0.2,0.7]（假定已经softmax归一化），实际分布为p=[1,0,0]那么交叉熵为-1\*log(0.1)=1，即只考虑真实类别的得分，那么求和符号可以去掉，由于p=1，那么此时的交叉熵也就是相对熵，计算的结果相同，前提是one-hot编码

如果p(x)不是独热编码而是类似于[0.95,0.3,0.2]的，那么相对熵和交叉熵不再等价，此时应当使用相对熵

对比多类SVM损失，如果未归一化q(x)=[0.6,-2.3,0.7]，实际类别是1，那么loss=max(0,-2.3-0.6+1)+max(0,1.9-0.6+1)=2.3

交叉熵需要先归一化，使用exp变为大于0，exp[q(x)]=[1.81,0.1,6.69],归一化得到[0.21,0.01,0.78],此时交叉熵为-log(0.21)=0.92

那么真正的区别在哪里呢，假设正确类为1，预测分布分别有3种情况，交叉熵损失为

[10,-2,3] = 0.0004

[10,9,9] = 0.2395

-np.log10(np.exp(10)/(np.exp(10)+np.exp(9)+np.exp(9)))

[10,-100,-100] = 1.5×10-48 损失非常小，这是因为得分差距越大，分母部分越接近分子，那么整个接近1取对数就是0

多类SVM，直观可以知道正确类10都比错误类高出至少1分，所以损失都是0，所以说交叉熵损失是希望不仅预测准确，而且正确类预测的概率远高于错误类，而不是预测正确就没有损失。

## 正则化项Regularization

与权重W有关，但是与图像数据无关，lambda是一个超参数，可以控制正则损失在总损失的比重，防止模型在训练集学习的太好

使用L1正则的经典线性回归模型叫Lasso回归

使用L2正则的模型叫Ridge回归

### L1正则

L1定义为所有元素绝对值的和，目的是为了权值绝对值最小化，趋向于使用更少的参数，其他为0，增加网络稀疏性

### L2正则

对大数值进行惩罚，喜欢分散权值，鼓励对所有维度特征进行利用，即使用更多的参数，但是这些参数接近于0，而不是强烈的依赖其中少数几维特征

例如样本[1,1,1,1]，分类器1的权值W1=[1,0,0,0]，分类器2的权值W2=[0,25,0.25,0.25,0.25]，两个分类器输出都是1，但是L2正则损失不同

L2损失定义为所有权值元素的平方和，目的是将参数的平方和最小化，那么R(W1)=1，R(W2)=0.25，所以分类器2更好

### L∞正则

### Dropout和早期停止

## 梯度优化器Optimizer

### 梯度下降法GD

核心公式：

缺点在于梯度的方向不容易指向最小值的方向，搜索效率低效

权重W的计算方式使用有限差分法，例如某个权重为

W = [0.34, 0.5, 0.7, 0.65, 0.8],此时的loss = 0.25546

然后第一项叠加很小的delta=0.001，其他项不动此时loss=0.25442

此时dW1 = (0.25442-0.25546) / 0.001 = -1.04

然后继续计算其他项得到

dW = [-1.04, dW2, dW3, ..., ]

While True :

data\_batch = sample\_training\_data(data,256) #小批量

weights\_grad = evaluate\_gardent(loss\_fun, data\_batch, weights)

weight += -lr \* weight\_grad

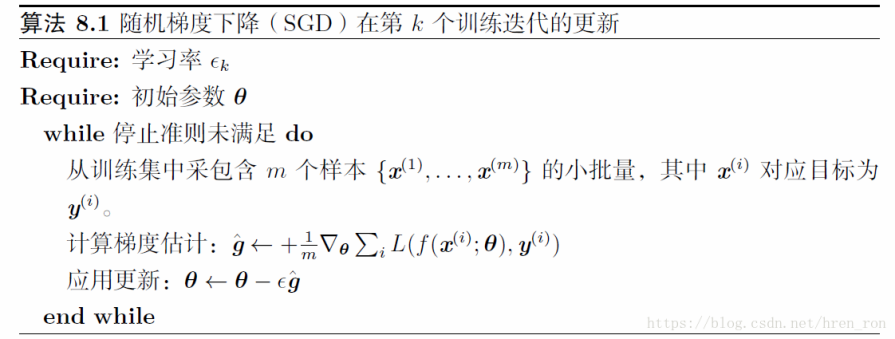
#### 批量梯度下降BGD

BGD每次更新参数使用所有的数据集，因此每次更新的方向是正确的，最终能够保证收敛于极值点（如果是凸函数，就是全局最优；如果是非凸函数，则是局部最优）

同样由于每次使用所有的数据集，因此更新时间较长，而且对于内存的消耗的较高，所以并不适合用于在线学习

#### 随机梯度下降SGD

随机梯度下降和BGD的方法类似，但是每次更新参数的时候并不是使用所有的训练集样本。SGD每次更新参数仅仅随机选择一个样本，因此会造成每次更新参数可能并不是朝着最优方向学习，学习的波动比较大



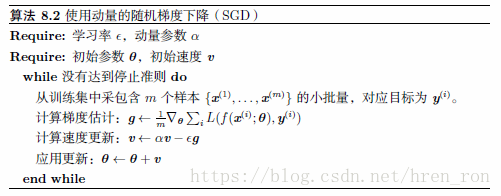
#### 小批量梯度下降MBGD

采用小批量的样本进行参数更新，每次从训练集中选择m个样本进行学习。这样可以避免使用所有的数据集，从而导致的学习速度过慢的问题；也可以避免每次只使用一个样本学习而导致的学习波动过大的问题

### 动量法Momentum

#### 标准动量

对SGD引入动量超参数α，一般的取值为0.5,0.9和0.99。和学习率一样，α也会随着时间不断调整，一般初始值是一个较小的值，随后慢慢变大



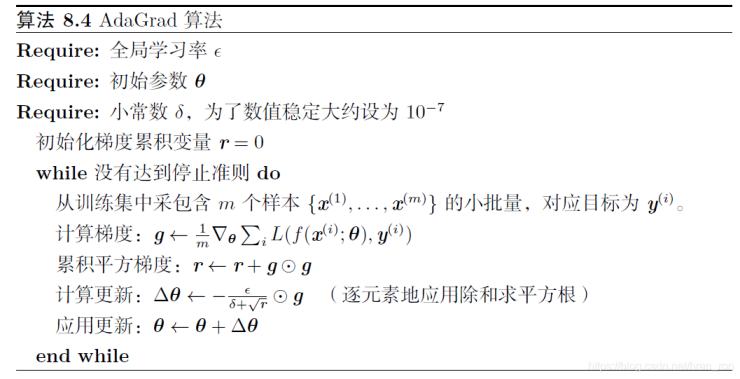
#### Nesterov动量

在标准动量的基础上添加一个校正因子λ，避免参数更新太快，提高灵敏度

### 自适应学习率法ALR

#### AdaGrad

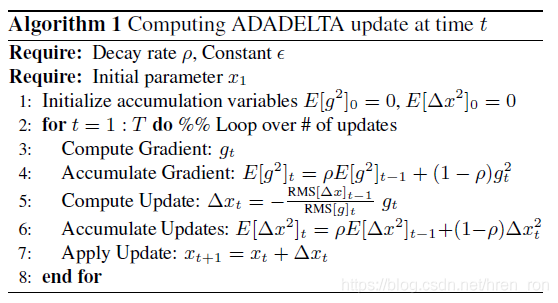
AdaGrad 算法可以独立地适应所有模型参数的学习率，缩放每个参数反比于其所有梯度历史平方值总和的平方根。具有损失最大偏导的参数相应地有一个快速下降的学习率，而具有小偏导的参数在学习率上有相对较小的下降



#### Adadelta

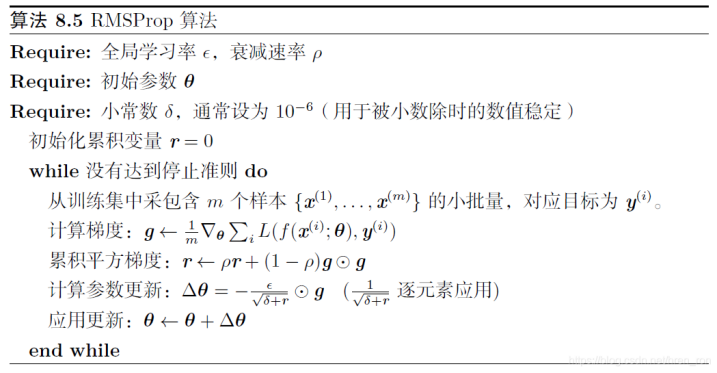
AdaDelta是对AdaGrad算法的改进，主要解决了两个方面的问题：

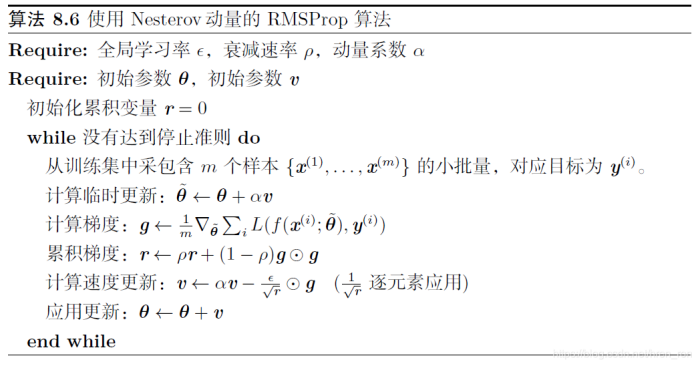
训练过程中学习率不断衰减，需要手工选择全局学习率



#### RMSProp

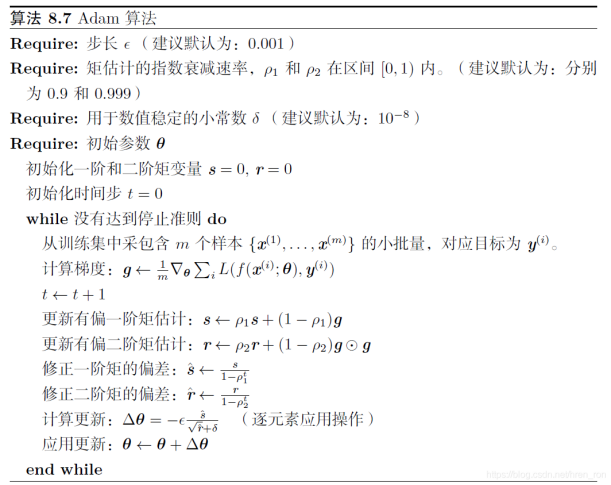
RMSProp 算法修改AdaGrad 以在非凸设定下效果更好，改变梯度积累为指数加权的移动平均。AdaGrad 旨在应用于凸问题时快速收敛。当应用于非凸函数训练神经网络时，学习轨迹可能穿过了很多不同的结构，最终到达一个局部是凸碗的区域





#### Adam

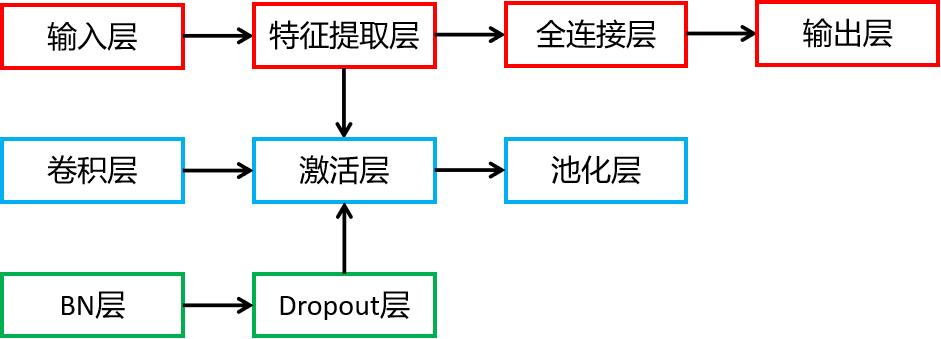
Adam 是另一种学习率自适应的优化算法，被看作结合RMSProp 和具有一些重要区别的动量的变种，引入两个超参数



## 卷积神经网络CNN

### 基本概念

本质上为4个组成部分，输入层、特征提取层、全连接层和输出层，特征提取层包括卷积层、激活层和池化层，还可以引入BN层和Dropout层



#### 卷积层

对于卷积层，由多个卷积滤波器组成，每个卷积滤波器一般都是3个通道，shape=(f,f,3)，和输入图像shape=(n,n,3)进行卷积，也就是每个通道对应元素相乘再求和，这些通道也要再进行一次求和填入到特征图的一个格中，这样随着滤波器的滑动结果就是一个二维特征图。相当于3通道的图片得到了2维特征图，即降低了一个维度，一个特征图可以看成是一幅图片的一个特征，可以使用多个滤波器得到多个特征，最后特征图的个数也就是下一层输出的第3维的维数。

输出层的shape=(o,o,c)，其中c已经可以确定不需要计算，只取决于上一层特征图的个数，即滤波器的个数。o的确定还取决于2个参数，即步长s和拓展p。

**一般的计算公式为**

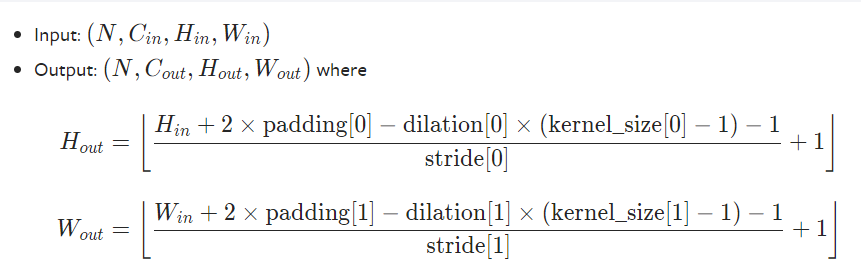
O =[ (N + 2P - F) / S + 1 ]

默认情况下P=0，S=1

常用的0拓展模式有2种，一种是默认的valid模式，也即是P=0，此时按照公式计算即可；另一种是same模式，P无需给定，会自动满足公式，输入输出的size不发生变化，也就是S=1且O=N时可以得到P=(F-1)/2，会自动填充0.

**完整的计算公式为：**

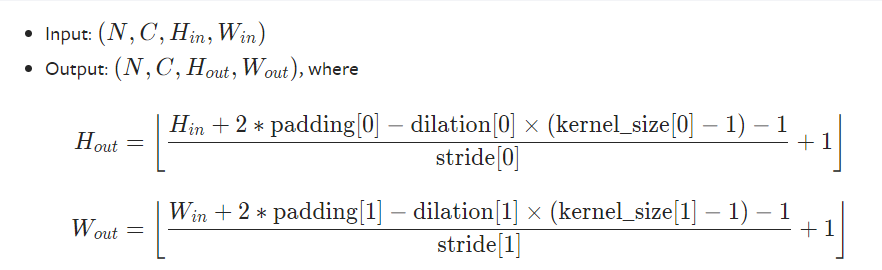
输入张量为（N，C，H，W）表示N个样本，通道数C，高度H，宽度W对应行和列的张量，输出（N，Cout，Hout，Wout），输入输出满足的关系



#### 池化层

池化方式由最大值池化和平均值池化，输入数据发生微小偏差时池化结果不变，因此对噪声具有鲁棒性，本质上思路和卷积层相同，样本数都不改变，区别在于对通道数不改变，步长为2，将形状shape降低一半，卷积层一般步长为1

最大池化有空洞卷积的情况，公式计算上和卷积层相同，有2种形式，但是平均池化没有空洞卷积的情况，只有简化形式，如下图



池化层 : 降低了各个特征图的维度，但可以保持大分重要的信息。池化层夹在连续的卷积层中间 （见书P240页）

压缩数据和参数的量，减小过拟合，池化层并没有参数，它只不过是把上层给它的结果做了一个下采样（数据压缩）

pooling在不同的通道depth上是分开执行的，也就是depth=5的话，pooling进行5次，产生5个池化后的矩阵

pool\_size(2,2)表示扫描区域大小2×2 , strides=(1,1)表示水平和竖直跳跃步长

池化操作是分开应用到各个通道特征图的，我们可以从64个输入图中得到64个输出图 即512\*512\*64 (pooling)----> 128\*128\*64

1) 最大值池化 : Max pooling 定义一个空间邻域如 2×2 的矩阵选择一个最大的元素作为被采样的点 那么4×4的图片可以压缩到 2×2

2) 平均值池化 :　Average pooling　取平均值

#### 激活层

常见的有RELU层、SOFTMAX层、SIGMOID层、TANH层

#### 全连接层

#### BN层

#### Dropout层

神经元以一定的概率停止工作

#### 卷积核类型

（1）检测边缘核

对于边缘检测分为4种方向【水平】【垂直】【西北东南】【东北西南】

对于【水平】可分为【左】和【右】，【垂直】可分为【上】【下】

【左】【下】掩膜可看成【右】和【上】的取反操作

一般的【右】掩膜具有的形式为 ：

[ cornor edge cornor ]

[ 0 0 0 ]

[ -cornor -edge -cornor]

一般的【上】掩膜具有的形式为 ：

[ -cornor 0 cornor ]

[ -edge 0 edge ]

[ -cornor 0 cornor ]

（2）轮廓线核

[outer, outer, outer],

[outer, inner, outer],

[outer, outer, outer]]

（3）浮雕

[[-diag, -iden, 0],

[-iden, iden, iden],

[0, iden, diag]]

（4）模糊

[[corner, edge, corner],

[edge, inner, edge],

[corner, edge, corner]]

（5）锐化

[[0, edge, 0],

[edge, inner, edge],

[0, edge, 0]]

（6）亮化

[[0, 0, 0],

[0, iden, 0],

[0, 0, 0]]

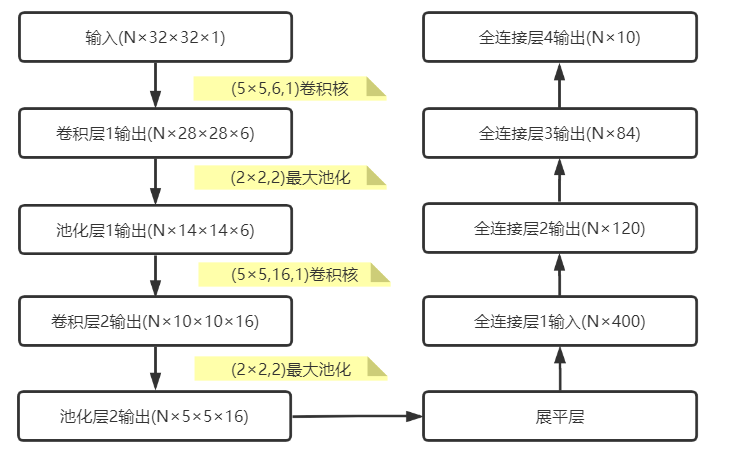
## 自编码器AutoEncoder

基于无监督学习的数据维度压缩和特征表示方法，无需对训练样本标记，常用于图像重构、聚类、降维。降维后的数据可以用于分类，去噪等

## 经典模型

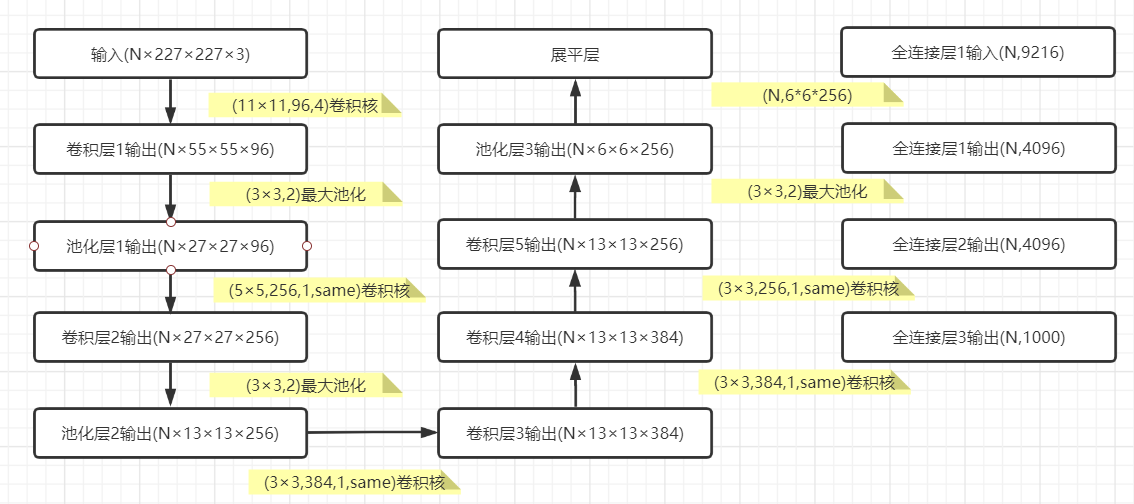
### 基础网络模型

#### LeNet



表示使用尺寸5×5，个数6，步长1，padding=valid的卷积层；尺寸2×2，步长2的池化层

#### AlexNet



第一个卷积层注意卷积步长为4；最后连续3个卷积层+最大池化，卷积层使用same保持形状不变；输出层使用softmax

#### VGGNet

##### VGG16

特点是全部使用3×3的滤波器大小和same填充，同时使用2×2的最大池化向下采样，步长2，依次使用深度为64、128、256、512的滤波器数量

13个Convd层+3个Dense层

输入层：224\*224\*3，暂时f=3

卷积层：

卷积I层：CBA-CBAPD

卷积层(64)➡BN层➡激活层(relu)

filters.shape=(3,3,64) p = (3-1)/2 = 1 ,f = 64

卷积层(64)➡BN层➡激活层(relu)➡最大池化层(2)➡随机失活层(0.2)

filters.shape=(3,3,64)，p=(f-1)/2=(64-1)/2=31 ,f = 64

此时图像形状为(112，112，64)

卷积II层：

卷积层(128)➡BN层➡激活层(relu) 此时图像形状为(112，112，128)

filters.shape=(3,3,128) p = (64-1)/2 = 31 ,f = 128

卷积层(128)➡BN层➡激活层(relu)➡最大池化层(2)➡随机失活层(0.2)

filters.shape=(3,3,128)，p=(f-1)/2=(128-1)/2=63 ,f = 128

此时图像形状为(56，56，128)

卷积III层：CBA\_CBA\_CBAPD

卷积层(256)➡BN层➡激活层(relu) 此时图像形状为(56，56，256)

filters.shape=(3,3,256) p = (256-1)/2 = 127 ,f = 256

卷积层(256)➡BN层➡激活层(relu)

filters.shape=(3,3,256) p = (256-1)/2 = 127 ,f = 256

卷积层(256)➡BN层➡激活层(relu)➡最大池化层(2)➡随机失活层(0.2)

filters.shape=(3,3,256)，p=(f-1)/2=(256-1)/2=127 ,f = 256

此时图像形状为(28，28，256)

卷积IV层：CBA\_CBA\_CBAPD

经过卷积后此时图像形状为(14，14，512)

池化后此时图像形状为(7，7，512)

卷积V层：和IV层完全相同的结构，此时图像形状为(3，3，512)

全连接层：然后进行了一些处理，2个全连接层展平向量为1\*1\*4096，再经过1个全连接得到1\*1\*1000，即输出层

#### InceptionNet

#### ResNet

### 语义分割网络

#### FCN

语义分割全卷积网络，输入和输出形状相同，通过反卷积（转置卷积）将池化+卷积的特征上采样，从而对像素分类

像素级别的图像分割，传统的网络ResNet、VGG等只能把图片整体进行分类，不能像素级别的分类。

语义分割可以将一张图片的多个人的像素点都归为一类，实例分割更强，还需要分为不同的人

#### U-Net

适用于医学图像和较小的训练集，仅有卷积层，没有全连接，不同于FCN的逐点相加，而是在通道维度上进行拼接融合

#### SegNet

借助自编码的思想，拥有编码器和解码器网络，编码器执行卷积和池化操作，并给出相应的最大池化索引，用于解码器进行上采样和卷积操作，最后每个像素通过softmax进行分类

#### DeepLab模型

用于语义分割，即像素层级对每个像素分配一个目标类，即语义标签，之后粗粒度转为细粒度模型

### 目标检测网络

#### 两阶段模型

需要先产生候选区域，再对候选区域对目标位置精修

##### R-CNN

是将CNN引入目标检测的开山之作

①输入图像使用Selective Search方法，生成1000-2000个ROI候选区域（即2000多个潜在物体类别的可能），对每个候选区域归一化统一尺寸（因为使用的网络是AlexNet，要求的图片尺寸固定），每个ROI都是224\*224×3，输出为4096的全连接向量

②2000×4096的特征矩阵送到SVM分类器，每个特征向量判断属于哪一类，只判断某个ROI是不是该物体，返回的是onehot向量

③回归网络修正ROI的位置

④使用非极大值抑制NMS方法对同类别的ROT合并，NMS的原理是得到每个矩形框的分数，两个矩形框的IOU超过指定阈值则保留分数大的矩形框

每个候选框都使用CNN提取特征，载将CNN提取的该候选框输出特征再送入SVM分类器，判断属于哪一类。得到的所有候选框知道类别以后需要进行合并，引入衡量矩形交叠情况的指标IOU，大于0.5时认为两个矩形基本相交

主要存在问题是：训练或者预测时2000个ROI都需要网络的前向过程提取特征，花费时间很长，然后SVM与回归都是和特征提取割裂开，也就是不会对前向提取特征过程产生反馈。

##### Fast R-CNN

RCNN，是先得到输入层的2000个候选框，在进行卷积，也就是卷积2000次；这里得到输入层候选框位置信息后不进行分别卷积，而是直接将图片卷积，卷积一次得到特征层，特征层上根据输入层的候选框位置和卷积公式即可得到特征层上的2000个ROI；然后分别经过池化，得到相同的尺寸，如6\*6\*256=9216，通过全连接层都是4096的向量；最后特征向量都送入softmax分类和L2回归，整体的损失函数为分类和回归的损失函数加权和，特征层的ROI可以因为分类和回归而进行调整

##### Faster R-CNN

两阶段方法的奠基性工作

提出的RPN（Region Proposal Networks）区域预测网络代替Selective Search算法，使得目标检测可以端到端，RPN本身是一个小的卷积网络。在Fast R-CNN的基础上，RPN放在最后一个卷积层之后，直接训练得到候选区域，RPN可以在特征图上滑动，每个滑动窗得到9个不同尺度、不同宽高的候选窗口，提取这9个窗口的特征用于目标分类和回归

对于分类，对于每个滑窗产生的anchor也就是候选窗口，都要计算anchor与真实标记矩形框的IOU，IOU大于0.7时认为含有物体，小于0.3不含有，介于中间则不参与网络训练的迭代

目标分类只需要区分候选框内是前景还是背景

#### 单阶段模型

无需候选区域，直接产生类别和位置坐标值

##### YOLO

图像分割为S×S的网格，每个网格通过深度卷积得到物体的所属类别，并在网格上生成B个边框，每个边框预测5个值，前4个是边框的位置和区域大小，第5个表示含有物体的概率和位置的准确程度，最后经过NMS（非极大抑制）过滤的到最后的预测框

##### SSD

##### Retina-Net

# 深度学习框架

## Tensorflow库

pip install tensorflow-cpu==2.6.0 -i <https://pypi.douban.com/simple/>

tensorflow CPU版本下载

pip install tensorflow-cpu==2.5.0 -i https://pypi.douban.com/simple/

pip install tensorflow-datasets==2.5.0 -i https://pypi.douban.com/simple/

pip install tensorflow-estimator==2.5.0 -i https://pypi.douban.com/simple/

pip install typing-extensions==3.7.4.3 # 2.5.0要求的，pylint会与之冲突

pip3 uninstall keras #解决kera和tf.kera冲突的问题

pip install --upgrade numpy

pip install -U numpy==1.19.2 # 版本不能冲突,tf不能用20版本的numpy

测试tensorflow下载是否成功

python

import tensorflow as tf

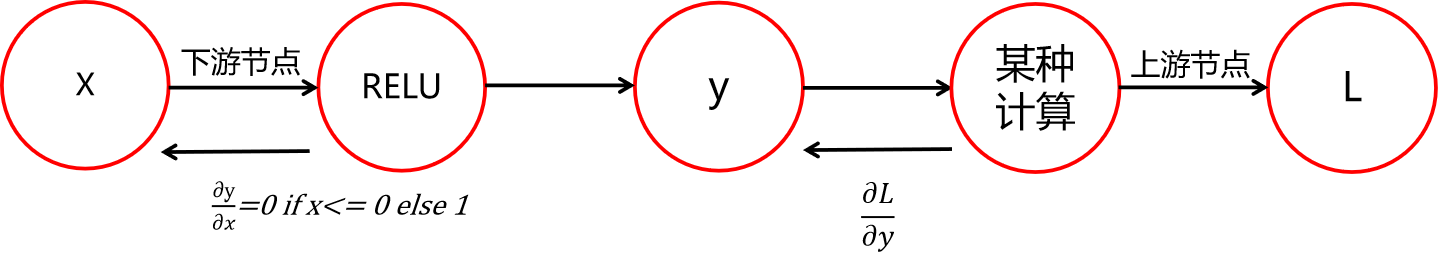
print(tf.\_\_version\_\_)

exit

### 计算图机制

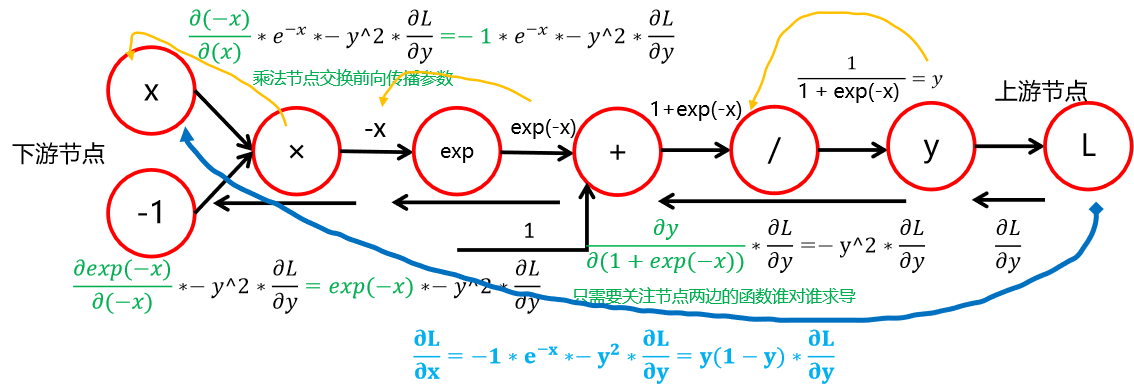
基本的计算图为加法和乘法符号。对于加法，如同减法是不影响梯度的；对于乘法，例如Z=XY，对变量X求导得到的是Y，所以乘法计算图是交换正向传播的值。

以RELU函数来说，计算图如下



上游L对x的梯度等于L对y的梯度与y对x的梯度乘积，梯度传递只关注节点运算是什么。L对y的梯度不知是何种运算则不进行化简，直接用L对y的偏导代替；y对x的运算取决于RELU运算符，该运算符满足图示的关系，非0即1，最终传递到下游的梯度非0即

再以Sigmoid函数为例，计算图如下



可以看出反向的梯度运算由4个节点梯度连乘得到，计算节点梯度一定要注意谁对谁求导，节点上游变量对节点下游变量进行整体求导，最后化简整理，尽量不出现除法、平方和指数运算

根据上述结果可以定义一个激活计算层

class Sigmoid:

def \_\_init\_\_(self):

self.out = None

def forward(self,x):

out = 1 / (1 + np.exp(-x))

self.out = out

return out

def backward(self,dout):

# dL/dx = y(1-y)dL/dy

dx = dout \* (1.0 - self.out)\* self.out

return dx

### 一般性训练流程

train/test/validdataset = tf.data.Dataset.from\_tensor\_slices()

# tf.keras.preprocessing.image\_dataset\_from\_directory

model = tf.keras.Sequential() # model.add()

lr\_sch = tf.keras.optimizers.schedules()

optimizer = tf.keras.optimizers(lr\_sch)

loss\_fn = tf.losses()

acu\_fn = tf.keras.metrics()

for i in range(epochs):

# training.....

for input , target in train\_dataset :

with tf.GradientTape() as tape: # 自动求导机制

output = model(input) # 模型预测的标签

loss = loss\_fn(output,target) # 计算损失

acu = acu\_fn(output,target) # 评估性能

grads = tape.gradient(loss, model.trainable\_variables) # 计算梯度

optimizer.apply\_gradients(zip(grads, model.trainable\_variables))# 梯度更新

# validing.....

for input , target in valid\_dataset :

# 模型输出、计算loss和评估性能acu 代码相同

model.predict(test\_dataset)

### nn模块

#### softmax

tf.nn.softmax(out, axis=1)

将输出张量按行归一化到概率值

#### dropout

"tf.nn.dropout(x, rate)

layer=tf.keras.layers.Dropout(rate,noise\_shape=None,seed=None);model.add(layer) # 创建层添加

model.add(layers.Dropout(rate=0.5))# 直接添加"

#### 交叉熵softmax\_cross\_entropy\_with\_logits

tf.nn.softmax\_cross\_entropy\_with\_logits(y\_true,y\_pred)

不需要先将y\_pred=tf.nn.softmax(y\_pred),此函数等价于归一化再计算交叉熵

### data.Dataset模块

#### 张量分割from\_tensor\_slices

①分割一维张量

dataset = tf.data.Dataset.from\_tensor\_slices([1, 2, 3])

list(dataset.as\_numpy\_iterator())

[1, 2, 3]

②分割二维张量

dataset = tf.data.Dataset.from\_tensor\_slices([[1, 2], [3, 4]])

list(dataset.as\_numpy\_iterator())

[array([1, 2], dtype=int32), array([3, 4], dtype=int32)]

③通用式

dataset = Dataset.from\_tensor\_slices((features, labels))

dataset = tf.data.Dataset.from\_tensor\_slices((X\_train,y\_train))

然后使用map,repeat,cache,shuffle,batch,prefetch生成数据启动器

train\_batch = dataset.repeat(1).shuffle(n\_samples).batch(batch\_size).prefetch(1)

之后在epoch内遍历即可

for x,y in train\_batch:

out = model(x)

....

#### 迭代器as\_numpy\_iterator

for step, (batch\_x\_train, batch\_y\_train) in enumerate

(dataset\_train.as\_numpy\_iterator()):

这是一个一个样本去迭代不是分批

#### 打包zip

tf.data.Dataset.zip((x\_train,y\_train))

打包数据和标签

#### 取take

Dataset.take(count)

取数据集前count个

count=-1或者大于数据集大小时表示全取

#### 跳过skip

Dataset.skip(count)

跳过数据集前count个

count=-1或者大于数据集大小时表示全部跳过

#### shuffle

Dataset.shuffle(buffer\_size)

打乱数据集,一般取数据集大小的缓冲区

list((dataset.shuffle(10).batch(3).repeat(2)).as\_numpy\_iterator())

先随机再块输出，特点是一次repeat里每个块数据不会重复

且数据集大小不能整除batchsize时最后1个块的size≤batch\_size

#### repeat

Dataset.repeat(count)

指定重复几轮，一般是随机-块输出的重复

list((dataset.batch(3).repeat(2).shuffle(2\*3)).as\_numpy\_iterator())

按顺序取3个直到最后然后重复，两组相同的数据拼接再去打乱输出指定块大小的batch，这里是对batch进行随机而非数据，而batch一开始是按顺序取好的，随机性不如前2个

#### batch

Dataset.batch(batch\_size)

指定取出的块大小

list((dataset.shuffle(10).repeat(2).batch(3)).as\_numpy\_iterator())

重复随机打乱两次数据，这两次进行组合再按块输出，块数据可能重复

最后1个块的size≤batch\_size

#### prefetch

### keras.models模块

#### 加载模型load\_model

load\_model(“cnn.h5”)

#### model内置方法

##### 建立模型build

需要指定input\_shape,如果不使用build就必须使用compile进行编译

##### 打印模型摘要summary

cnn.summary(line\_length=None, positions=None, print\_fn=None)

line\_length：打印行的总长度，用于适应终端口的大小

positions：每行中日志元素的相对或绝对位置。如果没有提供，默认值为[0.33、0.55、0.67、1.0]

print\_fn：它将在摘要的每一行调用，默认使用print，可以设置为自定义函数以捕获字符串摘要

##### 编译compile

cnn.compile(optimizer='rmsprop',loss=None,metrics=None,loss\_weights=None,weighted\_metrics=None,run\_eagerly=None,steps\_per\_execution=None,)

loss/metrics：为编译使用的损失函数核评价指标

详见<https://keras.io/zh/losses/> 和 <https://keras.io/zh/metrics/>

loss\_weights：损失加权，可以给出list

weighted\_metrics：待评估和加权的指标列表，训练和测试期间的样本权重

run\_eagerly：None

steps\_per\_execution：None

##### 训练fit

cnn.fit(x=None,y=None,batch\_size=None,epochs=1,verbose='auto',callbacks=None,validation\_split=0.,validation\_data=None,shuffle=True,class\_weight=None,sample\_weight=None,initial\_epoch=0,steps\_per\_epoch=None,validation\_steps=None,validation\_batch\_size=None,validation\_freq=1,max\_queue\_size=10,workers=1,use\_multiprocessing=False)

verbose： 'auto', 0, 1, or 2，0 = silent, 1 = progress bar, 2 = one line per epoch.

callbacks：回调，在keras.callbacks.Callback定义

validation\_split/validation\_data：交叉验证的比例和验证用的数据，(x\_val, y\_val, val\_sample\_weights)

class\_weight：用于对损失函数进行加权

sample\_weight：用于加权损失函数的训练样本

initial\_epoch：开始训练的时间（用于恢复以前的训练）

steps\_per\_epoch：步骤总数（样本批次），默认值None等于数据集中的样本数除以批量大小，传递无限重复的数据集时，必须指定每个参数的步骤

validation\_steps：每多少步进行验证

validation\_batch\_size：每次验证的batch大小

validation\_freq：验证频率，validation\_freq=2表示2个周期验证1次

max\_queue\_size：生成器队列的最大大小，默认10

workers：进程数

use\_multiprocessing：是否使用多进程，默认False

返回：A History object.History.history attribute is a record of training loss values and metrics values

##### 评估evaluate

一般用于测试数据

cnn.evaluate( x=None,y=None,batch\_size=None,verbose=1,sample\_weight=None,steps=None,callbacks=None,max\_queue\_size=10,workers=1,use\_multiprocessing=False,return\_dict=False)

steps：步骤总数（样本批次）

return\_dict：if True，loss and metric results are returned as a dict

##### 预测标签predict

cnn.predict(x,batch\_size=None,verbose=0,steps=None,callbacks=None,max\_queue\_size=10,workers=1,use\_multiprocessing=False)

##### 预测概率predict\_proba

cnn.predict\_proba(self, x, batch\_size=32, verbose=0)

##### 保存和加载模型参数save\_weights/load\_weights

model.save\_weights('weights.ckpt')

model.load\_weights('weights.ckpt')

保存和加载模型权重，需要创建相同形状的model

保存的仅仅是参数张量的数值，需要使用相同的网络结构才能够恢复网络状态

##### 保存save模型

model.save('model.h5')

del model

model = tf.keras.models.load\_model('model.h5')

以将模型的结构以及模型的参数保存到一个h5文件上

### keras.utils模块

#### 独热编码to\_categorical

to\_categorical(y, num\_classes=None, dtype='float32')

num\_classes：类别总数，使用None时使用(largest number in `y`) + 1

#### 绘制网络模型plot\_model

pip install pydot 和需要安装graphviz

tf.keras.utils.plot\_model(model=model,to\_file="model.png",show\_shapes=True,show\_layer\_names=True,dpi=400,expand\_nested=True,rankdir='TB')

expand\_nested：是否将嵌套模型展开为簇

rankdir：绘制格式，TB为竖直绘制，LR为水平

from tensorflow.keras.utils import plot\_model

from IPython.display import Image

plot\_model(net,to\_file="name.png",show\_shapes=True,show\_layer\_names=True)

Image(filename="name.png")

### keras.losses模块

#### 交叉熵CategoricalCrossentropy

loss=tf.keras.losses.CategoricalCrossentropy(from\_logits=False)

loss(y\_true, y\_pred).numpy()

计算标签和预测之间的交叉熵损失

需要先实例化再计算，或者直接使用函数

tf.keras.metrics.categorical\_crossentropy(y\_true,y\_pred,from\_logits=False)

①要求输入y\_true和y\_pred为独热编码,即样本数量\*分类数目的二维矩阵而非样本数\*1的标签编码向量，且y\_pred必须是浮点类型

②from\_logits=False表示默认输出的y\_pred为符合某种分布的概率，即在[0,1]之间不需要缩放，只需要归一化例如[0.2,0.6]-->[0.25,0.75]

③True时说明输出的y\_pred为[-inf,inf]表示须为未经过 Softmax 函数的变量

为了数值计算的稳定性，TensorFlow 中提供了一个统一的接口，将 Softmax 与交叉熵损失函数同时实现，避免单独使用 Softmax 函数与交叉熵损失函数

原理：

例1：

y\_true = [[1,0]]

y\_pred = [[0.9,0.1]]

0.105360545 = 1\*ln(0.9)+0\*ln(0.1)

使用函数和类计算的结果相同

对于二分类问题公式可以简化为-ln(P)

其中P是样本真实类上的概率

例2：

y\_true = [[1,0,0],[0,0,1]]

y\_pred = [[0.9,0.1,0.],[0,0.2,0.8]]

使用函数得到的结果为[0.10536055 0.22314353]，也就是分别进行计算

使用类得到的结果为0.16425204 表示取平均

#### 交叉熵categorical\_crossentropy

loss\_valid = tf.reduce\_mean(tf.keras.metrics.categorical\_crossentropy(y\_valid, out\_valid, from\_logits=False)).numpy()

需要指定y\_true和y\_pred，同时小写的计算返回的是批损失总和,所以必须使用reduce\_mean计算平均值

#### 稀疏交叉熵SparseCategoricalCrossentropy

loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=False)

loss(y\_true, y\_pred).numpy()

区别在于使用标签编码，即整数，但是要求在[0,3)内，浮点数只会被向上取整数，即y\_true=1.8也会被视作int(1)

也可以直接使用函数进行计算，不需要实例化tf.keras.losses.sparse\_categorical\_crossentropy(y\_true,y\_pred).numpy()

①计算归一化后的预测标签y\_pred，如果y\_pred已经满足概率分布和为1，实际上softmax变换之后结果不变

②若不满足概率和为1先进行归一化，实际上[x1,x2,..xn]-->[x1,x2,..,xn]/sum(x1+x2+...xn)

③计算ln(y\_pred)得到 n\_samples \* n\_classes 的对数矩阵

④计算y\_true的onehot编码得到y\_true\_onehot，可以利用to\_categorical进行计算

⑤计算矩阵的对应元素相乘 y\_true\_onehot \* ln(y\_pred)，然后按行取平均得到每个样本的交叉熵,最终再进行平均可得到标量值，也就是总的交叉熵

例子：

y\_true = [2,1,0,1.8]

y\_pred = [[0.1,0.6,0.3],[0,0.2,0.6],[0.2,0.2,0.4],[0.1,0.2,0.2]]

ln(y\_pred)=ln[[0.1,0.6,0.3],[0,0.25,0.75],

[0.25,0.25,0.5],[0.2,0.4,0.4]]

y\_true\_hot=[[0,0,1],[0,1,0],[1,0,0],[0,1,0]]

cross\_entropy = ln(0.3)+ln(0.25)+ln(0.25)+ln(0.4)=

rray([1.2039728 , 1.3862945 , 1.3862944 , 0.91629076])/4

=1.2232131（默认会求数组的平均）

使用reduction=tf.keras.losses.Reduction.NONE则返回数组不求平均

reduction=tf.keras.losses.Reduction.SUM返数组的和

#### 均方根误差mse

loss1 = tf.reduce\_mean(tf.keras.losses.mse(y\_train, output))

#### 均方根误差MSE

loss\_fn = tf.keras.losses.MSE()

loss2 = loss\_fn(y\_train, output)

### kera.metrics模块

#### 分类准确率CategoricalAccuracy

acu\_fn = tf.keras.metrics.CategoricalAccuracy()

实例化后使用acu = acu\_fn(y\_true,y\_pred)

#### 分类准确率categorical\_accuracy

acu\_valid = tf.reduce\_mean(tf.keras.metrics.categorical\_accuracy(y\_valid,out\_valid)).numpy()

返回的是批的总正确率，必须取平均

需要先实例化，再使用loss\_value =acu(y\_true,y\_pred)计算准确率

①acu.reset\_state()

②acu.result()

或者直接是使用方法tf.keras.metrics.categorical\_accuracy(y\_true, y\_pred)也可计算

"要求输入的y\_true是独热编码，计算方式是acu=count/total

①重置所有指标状态变量，在 epochs/steps 之间调用此函数

②返回最后一次更新的结果

from keras.utils.np\_utils import to\_categorical

y\_true = [0,1,2,3]

y\_true\_onehot = to\_categorical(y\_true, 4)

y\_pred = [[0.0,0.1,0,0],[0,0.8,0.1,0.1],[0.1,0.1,0.7,0.1],[0,0,0,1.]]

loss = tf.keras.metrics.CategoricalAccuracy()

loss1 = loss(y\_true\_onehot,y\_pred)

print(loss1.numpy())

loss2 = tf.keras.metrics.categorical\_accuracy(y\_true\_onehot, y\_pred)

print(loss2.numpy())"

### keras.optimizers模块

#### 应用梯度下降apply\_gradients

tf.keras.optimizers.Optimizer.apply\_gradients(grads\_and\_vars)

需要将梯度和变量使用zip打包，必须先对Optimizer实例化

opt = tf.keras.optimizers.SGD(learning\_rate=0.1,momentum=0.9) opt.apply\_gradients(grads\_and\_vars=grads\_and\_vars)

#### 梯度下降法SGD

tf.keras.optimizers.SGD(learning\_rate=0.1,momentum=0.9,nesterov=True,decay=1e-4)

momentum=0.9表示动量优化,一般设置0.9可比SGD更快

nesterov=True代表使用Nesterov加速梯度,比常规动量优化更快

对学习率采用幂调度使用decay,公式为lr=lr0/(1+t/s)^c,其中lr0为初始学习率,t为所在epoch数,s为经过的步骤数,c为幂数默认1,decay=1/s,s个步骤后下降到lr0/2,再s个步骤下降到lr0/3,...,以此类推先迅速下降然后越来越慢

#### 方均根RMSprop

tf.keras.optimizers.RMSprop(lr,rho=0.9)

克服了AdaGrad下降太快永远不会收敛到全局最优解的风险,rho为衰减率

RMSprop优化器,rho为历史或即将到来的梯度折扣因子

#### 自适应Adam

tf.keras.optimizers.Adam(lr,beta\_1=0.9,beta\_2=0.999,epsilon=1e-7)

自适应学习率方法，动量衰减超参数β1和β2,epsilon为平滑项防止零除

### keras.optimizers.schedules模块

#### 指数学习率ExponentialDecay

lr\_sch = tf.keras.optimizers.schedules.ExponentialDecay(

initial\_learning\_rate=lr, # 初始学习率

decay\_rate=0.98, # 0.1\*0.98^(step\*iters)

decay\_steps=iters, # 学习率每个周期/iter步下降1次

staircase = True) # 如果为真step/decay\_steps为整数除法,那么学习率曲线类似于阶梯型

opt = tf.keras.optimizers.SGD(lr\_sch) # 使用梯度下降法优化

#### 反时间比例学习率InverseTimeDecay

lr\_schedule = tf.keras.optimizers.schedules.InverseTimeDecay(

initial\_learning\_rate = 0.001,#初始学习率

decay\_steps=Iterations,

decay\_rate=1 )# 衰减率

# 反比例曲线模型的下降率

### keras.layers模块

#### Softmax层

tf.keras.layers.Softmax(axis=-1, \*\*kwargs)

构建softmax网络层，axis=-1默认表示最后一个维度进行归一化

#### 二维卷积Conv2D

Conv2D(input\_shape=(28,28,1),filters=64,kernel\_size=(3,3),strides=(1,1),data\_format=None,padding='valid',dilation\_rate=(1,1),groups=1,activation='relu',use\_bias=True,kernel\_initializer='glorot\_uniform',bias\_initializer='zeros',kernel\_regularizer=None,bias\_regularizer=None,activity\_regularizer=None,kernel\_constraint=None,bias\_constraint=None)

input\_shape=(28,28,1)：隐式参数，指明输入shape，一般在第一层说明

data\_format：数据格式，通道在前还是在后，可选channels\_last或channels\_first，默认在后，如果在pytorch或者opencv使用的在前

dilation\_rate：空洞卷积，默认为1，采样点之间没有距离，如果>1，那么输入输出形状的公式发生改变，完整的公式见池化层

groups：一个正整数，指定沿通道轴拆分输入的组数。每组分别与“过滤器/组”过滤器进行卷积。输出是沿通道轴的所有“组”结果的串联。输入通道和“过滤器”必须都能被“组”整除

kernel\_initializer：如何初始化权重和偏置，默认glorot\_uniform，类似的bias\_initializer默认使用zeros

kernel\_regularizer：正则化函数在“核”权矩阵中的应用，bias\_regularizer和activity\_regularizer类似

kernel\_constraint：应用于核矩阵的约束函数，类似的bias\_constraint

返回：activation(conv2d(inputs, kernel) + bias)

#### 最大池化层MaxPooling2D

MaxPooling2D(data\_format=None,pool\_size=(2,2),padding='valid',strides=None,input\_shape=None)

pool\_size：(pool\_height, pool\_width)

#### 全局平均池化GlobalAveragePooling2D

input\_shape = (2, 4, 5, 3)  
x = tf.random.normal(input\_shape) # 2个4×5高度3通道的图像  
y = tf.keras.layers.GlobalAveragePooling2D()(x)  
print(y.shape) #样本数和通道数不会变

data\_format：数据格式，默认channels\_last，可选channels\_first

Input\_shape：(batch\_size, rows, cols, channels)或者(batch\_size, channels, rows, cols)

Output shape：(batch\_size, channels)

#### 展平层Flatten

Flatten(data\_format=None)

#### 全连接层Dense

Dense(units=128,use\_bias=True,kernel\_initializer='glorot\_uniform',bias\_initializer='zeros',kernel\_regularizer=None,bias\_regularizer=None,activity\_regularizer=None,kernel\_constraint=None,bias\_constraint=None)

units：指明输出形状，神经元个数，其他参数见Conv2D

### keras.callacks模块

#### 监视板Tensorboard

Tensorboard(log\_dir='logs',histogram\_freq=0,write\_graph=True,write\_images=False,write\_steps\_per\_second=False,update\_freq='epoch',profile\_batch=2,embeddings\_freq=0,embeddings\_metadata=None,)

histogram\_freq：写入日志文件的epoch频率

write\_graph：是否输出模型图

然后在终端进入logs的文件夹，记得使用字符串：

cd C:\Users\chenb\jupyterLab

tensorboard --logdir="logs/mnist"

打开网站<http://localhost:6006/>

#### 模型参数回调ModelCheckpoint

tf.keras.callbacks.ModelCheckpoint(filepath=cbacks\_path+"\\ResNet50Model\_{epoch}.h5",save\_best\_only=True,save\_weights\_only=True,monitor=tf.keras.metrics.CategoricalAccuracy("accuracy").name)

要指定保存的路径和监控的指标

#### 早期停止EarlyStopping

early\_stop = tf.keras.callbacks.EarlyStopping

(monitor='val\_loss', patience=10,min\_delta=0)

使用早期停止技术，监控指标、最大改善周期、最小改善条件阈值

在model.fit(callbacks=[early\_stop])中使用

可以这样使用这个模块。



#### 自定义学习率LearningRateScheduler

def myScheduler(epoch): # 自定义学习率

    initial\_lr = 0.01

    end\_lr = 0.001

    # [(1+cos(epoch/epochs\*pi))/2 \* (1-end\_lr) + end\_lr] \* initial\_lr - lr

    rate = ((1 + math.cos(epoch \* math.pi / epochs)) / 2) \* (1 - end\_lr) + end\_lr  # cosine

    new\_lr = rate \* initial\_lr

    return new\_lr

学习率可以随周期变化，在compile的callbacks参数中使用即可。

model.compile(…,callbacks=tf.keras.callbacks.LearningRateScheduler(schedule=myScheduler)

如果是自定义训练过程，这样使用。

optimizer = tf.keras.optimizers.SGD(learning\_rate=0.01,momentum=0.9)

for epoch in range(epochs):

…

optimizer.learning\_rate = myScheduler(epoch) #每周期手动更

### summary模块

#### 事件记录器create\_file\_writer

"tf.summary.create\_file\_writer(

logdir, max\_queue=None, flush\_millis=None,

filename\_suffix=None, name=None)"

logdir：一个字符串，指定写入事件文件的目录

global\_path='C:\\Users\\chenbei\\VscodeItems\\myLeetCode\\myPleasure\\myClassification\\cTensorflow\\'

log\_dir=global\_path+"save\_logs\\"+datetime.datetime.now().strftime("%Y%m%d-%H%M%S")

train\_writer = tf.summary.create\_file\_writer(os.path.join(log\_dir, "train"))

val\_writer=tf.summary.create\_file\_writer(os.path.join(log\_dir,"val"))

然后与for epoch in range(epochs)同级别,也就是验证完之后使用以下代码

best\_val\_accuracy = 0;

for epoch in range(epochs):

…

    with train\_writer.as\_default():

        tf.summary.scalar("loss", train\_loss.result(), epoch)

        tf.summary.scalar("accuracy", train\_accuracy.result(), epoch)

    with val\_writer.as\_default():

        tf.summary.scalar("loss", val\_loss.result(), epoch)

        tf.summary.scalar("accuracy", val\_accuracy.result(), epoch)

    if val\_accuracy.result() > best\_val\_accuracy:

       best\_val\_accuracy = val\_accuracy.result()

   model.save\_weights(cbacks\_path+"\\model\_{}.ckpt".format(epoch), save\_format="tf")

#### scalar函数

scalar(name, data, step=None, description=None)

指定自定义的名称、数据(一般是loss或acu)、步数(一般是周期)和描述信息字符串，默认为空。

示例代码。

 test\_summary\_writer = tf.summary.create\_file\_writer('test/logdir')

 with test\_summary\_writer.as\_default():

      tf.summary.scalar('loss', 0.345, step=1)

      tf.summary.scalar('loss', 0.234, step=2)

  tf.summary.scalar('loss', 0.123, step=3)

#### image函数

#### histogram函数

#### SummaryWriter模块

### keras.Input模块

def Input(  # pylint: disable=invalid-name

    shape=None,

    batch\_size=None,

    name=None,

    dtype=None,

    sparse=None,

    tensor=None,

    ragged=None,

    type\_spec=None,

    \*\*kwargs):

等价于使用代码model = Model(input=[a, b], output=c)，a、b、c是张量。

Shape：元组，(32,)表示32长度的向量，可以是None。

Name：层的专有名称。

Sparse：一个布尔值，指定要创建的占位符是否稀疏。Ragged和sparrse只能有1个为真。如果'sparse'为False，稀疏张量仍然可以传递到输入中——它们将使用默认值0进行加密。

Tensor：可选的现有张量，以包装到“输入”层中。如果设置，图层将使用'tf'。而不是创建一个新的占位符张量。

Ragged：同sparse。

TypeSpec：用于创建输入占位符的对象。提供时，除name之外的所有其他参数都必须为None。

  x = Input(shape=(32,))

  y = Dense(16, activation='softmax')(x)

  model = Model(x, y)

 x = Input(shape=(32,))

  y = tf.square(x)  # This op will be treated like a layer

  model = Model(x, y)

### keras.utils模块

#### 独热编码to\_categorical

from keras.utils.np\_utils import to\_categorical

to\_categorical(int\_labels, num\_classes=None)

和tf.one\_hot功能一样

y\_true\_onehot = tf.one\_hot(y\_true,4)

将标签编码转为独热编码，返回array数组

onehot编码是根据位置给定元素1，如[2,1,0,0]

对应的[[0,0,1],[0,1,0],[1,0,0],[1,0,0]]

例子：

A = range(10)

A1 = to\_categorical(A, 10)

y\_true = [2,1,0,0]

y\_true\_onehot = to\_categorical(y\_true, 3)

### keras.datasets模块

#### mnist数据集

(train\_x,train\_y),(test\_x,test\_y)=mnist.load\_data()

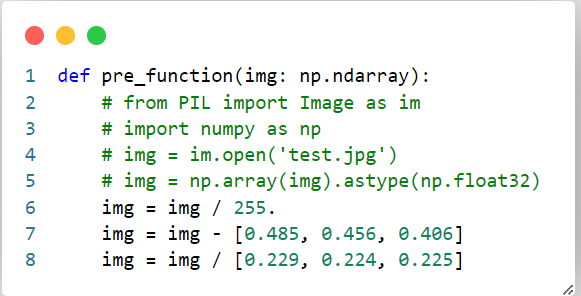
#### fashion\_mnist数据集

(train\_x,train\_y),(test\_x,test\_y)= fashion\_mnist.load\_data()

### keras.preprocessing.image模块

image\_generator=ImageDataGenerator(rescale=1./255,horizontal\_flip=True,validation\_split=0.2,preprocessing\_function=pre\_function)

使用方式，首先定义一个预先处理函数。



然后这样去使用，文件地址必须事先已经下载好数据。



### keras.applications模块

#### ResNet50模型

def ResNet50(include\_top=True,

             weights='imagenet',

             input\_tensor=None,

             input\_shape=None,

             pooling=None,

             classes=1000,

             \*\*kwargs):

include\_top：是否在网络顶部包含完全连接的层。

Weights：可以使用imagenet的预训练，也可以随机初始化还可以指定权重文件的路径。

input\_ tensor：可选Keras张量（即'layers.Input（）`）的输出，用作模型的图像输入。

include\_shape：可选形状元组，仅在“include\_top”为False时指定（否则输入形状必须为“（224，224，3）”（使用“channels\_last”数据格式）或“（3，224，224）”（使用“channels\_first”数据格式）。它应该有3个输入通道。

pooling：当“include\_top”为“False”时，用于特征提取的可选池模式。使用None，这意味着模型的输出将是最后一个卷积层的4D张量输出。Avg说明全局平均池将应用于最后一个卷积层的输出，因此模型的输出将是一个二维张量。Max说明使用最大池化。

#### 预处理函数resnet50.preprocess\_input

@keras\_export('keras.applications.resnet50.preprocess\_input',

              'keras.applications.resnet.preprocess\_input')

def preprocess\_input(x, data\_format=None):

  return imagenet\_utils.preprocess\_input(

      x, data\_format=data\_format, mode='caffe')

内部调用了imagenet\_utils.preprocess\_input函数，对编码一批图像的张量或Numpy数组进行预处理，处理数据格式,caffe,tf,torch是不一样的。

  if mode not in {'caffe', 'tf', 'torch'}:

    raise ValueError('Unknown mode ' + str(mode))

  if data\_format is None:

    data\_format = backend.image\_data\_format()

  elif data\_format not in {'channels\_first', 'channels\_last'}:

    raise ValueError('Unknown data\_format ' + str(data\_format))

  if isinstance(x, np.ndarray):

    return \_preprocess\_numpy\_input(

        x, data\_format=data\_format, mode=mode)

  else:

    return \_preprocess\_symbolic\_input(

        x, data\_format=data\_format, mode=mode)

### 常用函数

#### 自动求导GradientTape

with tf.GradientTape() as tape:（必须包括在梯度上下文中）

grads = tape.gradient(loss1, model.trainable\_variables)

#### 独热编码one\_hot

y\_true\_onehot = tf.one\_hot(y\_true,4)

转为独热编码,和to\_categorical区别返回的是tensor类型

#### 平均reduce\_mean

loss = tf.reduce\_mean(loss)

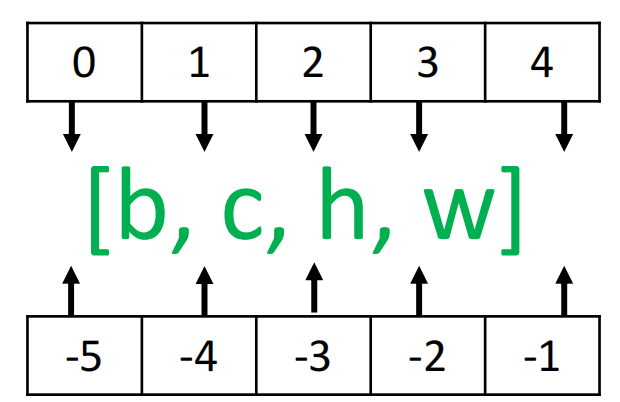
返回标量1个值

#### 维度拓展expand\_dims

tf.expand\_dims(x, axis)

指定的 axis 轴前可以插入一个新的维度

tf.expand\_dims(x,axis=0)(1, 28, 28)=>(b,1,28,28)



上图就表示在轴0前再插入1个轴,表示批样本数

#### 维度削减squeeze

tf.squeeze(x, axis)

不指定维度参数 axis

会默认删除所有长度为 1 的维度

(1,28,28,1)-->(28,28)

#### 最值索引argmax/argmin

argmax(input, axis=None, output\_type=dtypes.int64, name=None)

#### 强制转换cast

例如tf.cast(label,tf.int32))可以把标签强制转为整型

#### 张量转换convert\_to\_tensor

X\_train\_pca = tf.convert\_to\_tensor(X\_train\_pca,dtype=tf.float32)

## Pytorch库

安装方法：pip3 install torch torchvision torchaudio

网址：<https://pytorch.org/>

注：在Pytorch机制中，输入的核和图片张量必须都是torch.float32类型，不能使用uint8或者float64，另外有标签的时候也要转化为有符号64位整型，即torch.int64类型，可以使用tortch.to方法转换数据类型

### 自动求导机制

生成张量时，可以指定参数requires\_grad=True表明该张量是可导的

import torch

x = torch.tensor([[1.,2.],[3.,4.]],requires\_grad=True)

y = torch.sum(x\*\*2+2\*x+1)

y.backward() # y对x反向传播的梯度矩阵放在x中

result1 = x.grad # x 具有梯度信息

result2 = y.grad # y None

### 一般性训练流程

dataset =from\_numpy+ TensorDataset/torchvision.datasets/ImageFolder

model = super(torch.nn.Module)

optimizer = torch.optim(model.parameters(),...)

scheduler = torch.optim.lr\_scheduler.(optimizer,...)

# 评估函数需要自己引入或者定义

# 可以借助sklearn.metrics.explained\_variance\_score（回归问题）

# accuracy\_score是分类问题

acu\_fn = ......

loss\_fn = torch.nn()

for i in range(epochs):

for input , target in dataset :

optimizer.zero\_grad() # 梯度清零

output = model(input) # 模型预测的标签

loss = loss\_fn(output,target) # 计算损失

acu = acu\_fn(output,target) # 计算得分

# 打印时对于要求求导的使用loss.detach().numpy()

loss.backward() # 自动求导机制

optimizer.step() # 梯度更新

scheduler.step()

### nn模块

#### 初始化参数init

常用的有4个，均匀分布、正态分布、常值和全1，可以对某个层的权重参数初始化，torch.nn.init.normal\_(layer.weight,mean=0,std=1),常值可以用于对偏置填充，即torch.nn.init.constant\_(layer.bias,val=0.1)

均匀分布

torch.nn.init.uniform\_(tensor, a=0.0, b=1.0)

正态分布

torch.nn.init.normal\_(tensor, mean=0.0, std=1.0)

常值

torch.nn.init.constant\_(tensor, val)

全1

torch.nn.init.ones\_(tensor)

#### 网络模型类Module

方法：

① 获取前向输出的方法

register\_forward\_hook (hook)

② 获取反向计算的方法

register\_forward\_hook(hook)

属性：

#### 二维卷积Convd2d

conv2d=torch.nn.Conv2d(in\_channels, out\_channels, kernel\_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding\_mode='zeros', device=None, dtype=None)

in\_channels: int, 输入图像通道数

out\_channels: int,输出通道数

kernel\_size: \_size\_2\_t,卷积核尺寸

stride: \_size\_2\_t = 1,卷积步长

padding: Union[str, \_size\_2\_t] = 0,填充

padding='valid' is the same as no padding. padding='same' pads the input so the output

dilation: \_size\_2\_t = 1,卷积核元素之间的步长，用于空洞卷积

groups: int = 1,输入通道到输出通道的阻塞连接数

bias: bool = True,默认添加偏置

返回一个定义好的卷积层类，具备属性weight和bias，可以进一步查看data得到它们使用的卷积核，data的长度和kernel个数相同，也就是out\_channels，可以指定核为自定义的核，否则默认高斯分布初始化权重

example：

im = skimage.img\_as\_float(im)

im = torch.from\_numpy(im).reshape((1,1,im.shape[0],im.shape[1]))

im = im.to(torch.float32)

kernel\_size = 5

kernel\_1 = torch.ones(size=(kernel\_size,kernel\_size),dtype=torch.float32)\*-1

kernel\_2 = kernel\_1

kernel\_1[2,2]=25 # 轮廓线核

kernel\_2[0,0],kernel\_2[0,-1],kernel\_2[-1,0],kernel\_2[-1,-1] = [0,0,0,0]

kernel\_2[2,2]=25 # 锐化核

kernel\_1 = kernel\_1.reshape((1,1,kernel\_size,kernel\_size))

kernel\_2 = kernel\_2.reshape((1,1,kernel\_size,kernel\_size))

conv2d = torch.nn.Conv2d(in\_channels=1,out\_channels=2,kernel\_size=(kernel\_size,kernel\_size),bias=False)

conv2d.weight.data[0] ,conv2d.weight.data[1] = kernel\_1,kernel\_2

im\_re = conv2d(im)

im\_re = im\_re.squeeze()

# 返回的不是图片的3个通道，而是特征图，这里得到2个特征图

plot\_im1\_im2(im\_re.data.numpy()[0],im\_re.data.numpy()[1])

#### 二维池化MaxPool2d

torch.nn.MaxPool2d(kernel\_size,stride=None,padding=0,dilation=1,return\_indices=False, ceil\_mode=False)

其他参数与上述相同含义

return\_indices=False，表示默认不返回最大值的索引，True一般用于池化逆操作，也就是torch.nn.MaxUnpool2d()函数

ceil\_code：默认向下取整，True表示向上取整

注：池化层不改变输出的通道数、样本数，只改变高度和宽度，关于池化层的具体情况见55.55.5.15.5.1.2池化层的说明，公式与卷积层的公式相同

还有平均池化、自适应平均池化、自适应最大池化

torch.nn.AdaptiveAvgPool2d(output\_size)

output\_size：指定输出尺寸，是2d的，输出会自动拼接样本数N和通道数C，卷积层必须指定4d输入，这是参数上的略微不同

torch.nn.AvgPool2d(kernel\_size, stride=None, padding=0, ceil\_mode=False, count\_include\_pad=True, divisor\_override=None)

count\_include\_pad：默认True，会将padding参数考虑在内计算

divisor\_override：如果指定它将用作除数，否则将使用内核大小，一般忽略

torch.nn.AdaptiveMaxPool2d(output\_size, return\_indices=False)

#### 激活函数层

常见的有

torch.nn.ReLU()

torch.nn.Tanh()

torch.nn.Sigmoid()

torch.nn.Softmax()

torch.nn.Softplus()

#### 损失函数

L1Loss() 平均绝对值误差 适用于回归

L1Loss(size\_average=None, reduce=None, reduction='mean')

SmoothL1Loss() 平滑L1损失 适用于回归

SmoothL1Loss(size\_average=None,reduce=None,reduction='mean',beta=1.0)

MSELoss() 均方误差损失 适用于回归

MSELoss(size\_average=None, reduce=None, reduction='mean')

size\_average默认True，计算的损失为每个batch的均值，否则为每个batch损失的和，reduce默认True，此时计算的损失根据size\_average参数设定，计算均值或者和，这两个参数以后会弃用，改用reduction来代替

reduction可以取mean、sum或者none

CrossEntropyLoss() 交叉熵损失，适用于多分类，这里pytorch不支持onehot格式，与tensorflow不同，对应交叉熵损失，稀疏交叉熵支持onehot

CrossEntropyLoss(weight=None,size\_average=None,ignore\_index=-100,reduce=None, reduction='mean')

ignore\_index表示忽略指定的对输入梯度没有贡献的目标值

weight为1维张量，包含n个元素，代表n类的权重，训练样本不平衡时采用

NLLLoss() 负对数似然损失函数 适用于多分类

NLLLoss(weight=None,size\_average=None,ignore\_index=-100, reduce=None, reduction='mean')

NLLLoss2d() 图片负对数似然损失函数 适用于图像分割

KLDivLoss() KL散度损失 适用于回归

KLDivLoss(size\_average=None,reduce=None,reduction='mean',log\_target=False)

BCELoss() 二分类交叉熵损失 适用于二分类

MarginRankingLoss 相似度损失

MultiLabelMarginLoss 多标签分类损失 适用于多标签分类

SoftMarginLoss() 多标签二分类损失 多标签二分类

#### 线性层Linear

全连接层可以看成线性层与激活函数层的叠加

torch.nn.Linear(in\_features, out\_features, bias=True, device=None, dtype=None)

需要指定每个输入/输出样本的特征数量

输入(N,in\_features)的张量，返回(N,out\_features)的张量

#### 二维逆池化MaxUnpool2d/AvgUnpool2d

######

### optim模块

#### lr\_scheduler模块

用于改变学习率的模块

LambdaLR(optimizer, lr\_lambda, last\_epoch=-1, verbose=False)

new\_lr=λ×initial\_lr

StepLR(optimizer, step\_size, gamma=0.1, last\_epoch=-1, verbose=False)

new\_lr=initial\_lr×γ ^(epoch//step\_size)

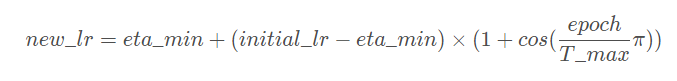
MultiStepLR(optimizer, milestones, gamma=0.1, last\_epoch=-1, verbose=False)

new\_lr=initial\_lr×γ ^[bisect\_right(milestones,epoch)]

ExponentialLR(optimizer, gamma, last\_epoch=-1, verbose=False)

new\_lr=initial\_lr×γ ^epoch

CosineAnnealingLR(optimizer,T\_max,eta\_min=0,last\_epoch=-1, verbose=False)



last\_epoch: 最后一个epoch的index，如果是训练了很多个epoch后中断了，继续训练，这个值就等于加载的模型的epoch。默认为-1表示从头开始训练，即从epoch=1开始

#### SGD

SGD(params,lr=required,momentum=0,dampening=0,weight\_decay=0,nesterov=False)

params：用于优化或定义参数组的参数表

optim.SGD([{'params':model.base.parameters()},{'params':model.classifier.parameters(), 'lr': 1e-3}], lr=1e-2, momentum=0.9)

或者optim.SGD(model.parameters(), lr=0.01, momentum=0.9)

momentum：标准动量，一般取0.9，或者0.99

weight\_decay：权重衰减（L2惩罚），默认为0

dampening：动量阻尼（默认值：0）

nesterov：是否使用nesterov的动量法

#### Adam

torch.optim.Adam(params,lr=0.001,betas=(0.9,0.999),eps=1e-08,weight\_decay=0, amsgrad=False)

params：待优化参数的iterable或者定义了参数组的dict列表，通常为model.parameters()，model是一个定义好的模型

指定参数组时，若模型有2层，可以对2个层分别使用不同的学习率

[ {"params":model.hidden.parameters(),"lr":0.0001},

{"params":model.regression.parameters(),"lr":0.01}]

weight\_decay：权重衰减（L2惩罚），默认为0

#### Adadelta

#### Adagrad

#### Adamax

#### ASGD

平均梯度下降

#### LBFGS

#### RMSprop

#### Rprop

弹性反向传播

### utils.data模块

用于常用的数据预处理的操作，如读取、切分和准备

#### 数据张量处理TensorDataset

orch.utils.data.TensorDataset(\*tensors)

train\_data = torchdata.TensorDataset(train\_xt,train\_yt)

可以将两个张量整合到一起

#### 连接数据集ConcatDataset

torch.utils.data.ConcatDataset(datasets)

数据集作为多个数据集的串联，此类用于组装不同的现有数据集。

#### 获取数据子集Subset

torch.utils.data.Subset(dataset, indices)

指定索引处的数据集子集

#### 数据迭代器DataLoader

DataLoader(dataset, batch\_size=1, shuffle=False, sampler=None,

batch\_sampler=None, num\_workers=0, collate\_fn=None,

pin\_memory=False, drop\_last=False, timeout=0,

worker\_init\_fn=None, \*, prefetch\_factor=2,

persistent\_workers=False)

dataset主要有3种方式：

第一种是直接加载numpy数据，使用from\_numpy和TensorDataset对数据和标签进行组合；

第二种是从torchvision.datasets下载数据集，得到的数据集需要先类型转换、归一化和维度拓展处理；

第三种是直接从本地文件夹加载，需要使用ImageFolder类定义数据加载器

sampler：自定义的洗牌方式，指定的话shuffle不能指定

batch\_sampler（sampler或Iterable，可选）：与sampler类似，但一次返回一批索引

num\_workers（int，可选）–用于数据加载的子进程数。0表示将在主进程中加载数据。（默认值：0）

其他参数一般不需使用

返回一个迭代器，迭代器每次可以输出批样本数据和批样本标签

类似于train\_dataset=tensorflow的from\_tensor\_slices.shuffle.batch.repeat的数据流程，for step, (x,y) in enumerate(train\_dataset)

train\_loader=torchdata.DataLoader(dataset=train\_data,batch\_size=64,shuffle=True,num\_workers=2)

for step , (bx,by) in enumerate(train\_loader):

print(bx.shape,by.shape) # torch.Size([64, 13]) torch.Size([64])

还有一种方法，将启动器转为可迭代的。

#### 数据加载器ImageFolder

train\_data=ImageFolder(path,transform=transform,target\_transform=None)

可以读取的数据集格式是一个文件下的多个子文件夹，每个子文件夹代表1类图像。

path = "C:\\Users\chenb\\.torch\\flower\_photos\\"

# 转换之前数据是uint8类型和HWC的格式，各个图片大小不同

transform=Compose([torchvisiontrans.Scale((224,224)),ToTensor(),Normalize(mean=[0,0,0],std=[1,1,1],inplace=False),])

# 使用ToTensor()此时才会映射为float.32，标签int64

train\_data = ImageFolder(path,transform=transform)

# a = plt.imread(train\_data.imgs[0][0])

# print(a.shape,a.dtype)# (263, 320, 3) uint8

train\_loader=DataLoader(dataset=train\_data,batch\_size=64,shuffle=True,num\_workers=2)

for step,(bx,by) in enumerate(train\_loader\_):

print(step,bx.shape,bx.dtype,by.shape,by.dtype,torch.max(bx),torch.min(bx))

# torch.Size([22, 3, 224, 224]) torch.float32 torch.Size([22]) torch.int64 tensor(1.) tensor(0.)

#### 随机划分random\_spilit

torch.utils.data.random\_split(dataset, lengths, generator=<torch.\_C.Generator object>)

将数据集随机拆分为给定长度的非重叠新数据集。可选地固定发生器，以获得可重复的结果

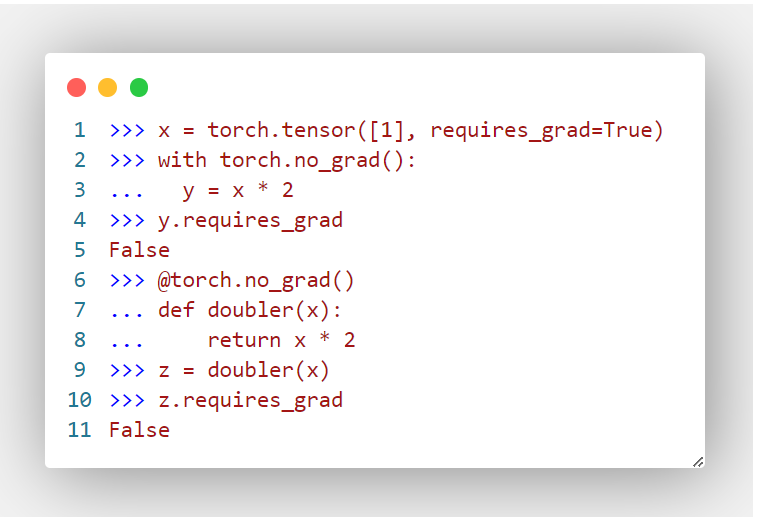
random\_split(range(10), [3, 7], generator=torch.Generator().manual\_seed(42))

### 常用函数

##### no\_grad取消梯度计算

一个禁用梯度计算的上下文管理器。

使用方法如截图，x是一个要求梯度计算的，y可以不是。或者使用装饰器修饰函数也是可以的。





##### 重塑reshape

x = tf.reshape(x,(-1,x.shape[1],x.shape[2]))

##### 散点scatter\_

此函数主要用于独热编码生成使用，其他功能少用

scatter\_(dim=1,index=label,value=1)

dim：转换的维度，一般是二维的

value：填充的值，一般取1

index：转变的标签，label是一个列向量的标签，例如[0,1,2,3],

可以使用label.reshape((-1,1))或者label.unsqueeze(dim=1)调整为列向量

可以构造一个(m,n)的全0矩阵,m是样本数,n是类别数,注意n的长度必须大于等于index的长度，例如label是5分类的，构造的矩阵列数必须大于等于5，然后使用torch.zeros((m,n)).scatter\_(dim=1,index=label,value=1)

##### 看作view

x = torch.linspace(-6,6,100).view((-1,2))

类似于reshape的用法，可以把数组的形状改变

##### 维度扩展stack

指定维度拓展

stack((t1,t2),dim=0)

t1和t2必须相同维度

##### 拼接cat

cat((t1,t2),dim=0)

指定的维度拼接，如(2,4)和(4,4)可以拼接成(6,4)

##### 保存和加载模型save/load

# 保存整个模型

path = os.getcwd()+"\\Pytorch深度学习入门与实战\\保存模型.pkl"

# torch.save(model\_1,path)

# model = torch.load(path) # 加载模型

# 只保存模型参数

torch.save(model\_1.state\_dict(),path)

model\_param = torch.load(path)

print(model\_param)

##### 设置类型set\_default\_tensor\_type

torch.set\_default\_tensor\_type(t=torch.DoubleTensor)

也可以使用函数，函数带入的是类型而非参数

torch.set\_default\_dtype(d=torch.float32)

##### 元素数量numel

tortch.numel()

##### 生成张量from\_numpy/as\_tentor

##### 随机种子manual\_seed

##### 随机排序randperm

tortch.randperm(n)

可将0-n的整数随机排列输出

##### 相同形状resize\_as\_

A.resize\_as\_(B)

##### 插入维度unsqueeze

torch.squeeze(input, dim)

指定位置插入维度

##### 删除维度squeeze

torch.squeeze(input, dim=int)

默认删除全部维度大小为1的维度，可以指定删除某个维度大小为1的维度

##### 拓展维度expand/expand\_as/repeat

torch.expand(batch\_size, bsz)

张量元素比较

##### 元素相等eq/equal

tortch.eq() 逐个比较元素是否相等

tortch.equal() 判断两个张量shape和元素是否相等

##### 裁剪函数clamp

torch.clamp(input,min,max,out)

torch.clamp\_max(input,max)

torch.clamp\_min(input,min)

##### 逆和转置inverse/t

##### 排序sort

torch.sort(input dim, descending=False)

可以返回排序后的结果和排序位置,默认升序

##### 元素第k大和第k小

torch.topk(input, k, dim=int)

torch.kthvalue(input, k, dim, keepdim)

返回张量指定维度取值为第k大和第k小的数值和数值所在位置

##### 比较大小max/min

values,indices = torch.max(temp\_y[-1],dim=1)

返回值和索引 dim=1表示在第2个维度去比较大小 也就是某行的列元素之间比较，可用torch.argmax替代

torch.return\_types.max(

values=tensor([2.3218, 6.7678]),

indices=tensor([3, 8]))

取出第2个值indices就是最大值所处的位置

准确率的计算：

outputs = model(val\_image)

predict\_y = torch.max(outputs, dim=1)[1] # max在第2个维度比较,某行列元素之间,可用argmax代替

accuracy = torch.eq(predict\_y, val\_label).sum().item() / val\_label.size(0)

##### 返回最值索引argmax/argmin

indices\_ = torch.argmax(temp\_y[-1],dim=1)

##### 统计量

mean、sum、cumsum、cumprod、median、std

## Torchvision库

### transform模块

#### 组合变换Compose

Compose([])

trans = Compose([ToTensor(),Scale((224,224)),RandomHorizontalFlip(p=0.5)])

要注意的是：输入为张量时很多变换不能使用，所以建议ToTensor()放在最后使用

#### 转为张量ToTensor

将numpy的ndarray或PIL.Image读的图片转换成形状为(C,H, W)的Tensor格式，且/255归一化到[0,1.0]之间.

注：numpy中的元素必须时uint类型时才会将[0，255]归一化到[0,1.0]之间,否则不作映射

transform= torchvisiontrans.ToTensor()

eg：

a = np.array([[[255,255,127],[255,255,127]],

[[255,255,127],[255,255,127]],

[[255,255,127],[255,255,127]]]).astype(np.uint8)

trans = torchvisiontrans.ToTensor()

b = trans(a)

print(a.shape,b.shape,a.dtype,b.dtype)

out：(3, 2, 3) torch.Size([3, 3, 2]) uint8 torch.float32

数组格式是3行2列3个通道，对应应当是高3宽2通道3，转换为3通道3行2列的张量

#### 缩放变换Scale

Scale((224,224))

缩放变换现在会有使用警告不推荐使用，改用Resize更好

#### 调整大小Resize

Resize(size,interpolation=InterpolationMode.BILINEAR,max\_size=None,antialias=None)

size：(sequence or int)。sequence like(h, w)；如果是int，图像的较小边缘将与此数字匹配，如果高度大于宽度，图片会缩放到(size \* height / width, size)

interpolation：插值支持的枚举类型可在

torchvision.transforms.InterpolationMode中寻找

默认InterpolationMode.BILINEAR，如果输入是张量，只支持

InterpolationMode.NEAREST，

InterpolationMode.BILINEAR，

和InterpolationMode.BICUBIC

max\_size：只在size=int时使用，允许的较长边缘的最大值，如果最大长度比size给出的大，则以size为主，否则以max\_size为主

antialias：输入的是PIL对象则忽略，如果输入是张量且InterpolationMode.BILINEAR时才能设为True，其它情况默认False

#### 随机切割指定大小图像RandomSizedCrop

RandomSizedCrop(224)

#### 水平翻转RandomHorizontalFlip

RandomHorizontalFlip(p=0.5)

#### 标准化Normalize

使用平均值和标准偏差对浮点张量图像进行规格化，此转换不支持PIL图像。

Normalize(mean,std,inplace=False)

输入必须是(C, H, W)或者(B,C, H, W)

mean和std必须给定每个通道的取值，是1×3的列表[]

### datasets模块

#### MNIST

torchvision.datasets.MNIST(root,train,download,transform,target\_transform)

root：MNIST所在的文件夹

train：如果为True，则从training.pt创建数据集，否则从test.pt创建数据集

download：从网络下载数据集，已存在则不下载

transform：接受PIL图像并返回已转换版本的函数/转换

target\_transform：接受目标并对其进行转换的函数/转换

transform是transform模块中的类

#### FashionMNIST

类似于MNIST

torchvision.datasets.FASHIONMNIST(root,train,download,transform,target\_transform)

#### CIFAR10

类似MNIST

train\_set=torchvision.datasets.CIFAR10(root,train,download,transform)

### models模块

#### Alexnet\_5

alexnet = torchvision.models.alexnet(pretrained=False,progress=True)

#### vgg16

vgg16 = torchvision.models.vgg16(pretrained=True)

vgg = vgg16.features # 只使用特征提取层，不包括全连接

for param in vgg.parameters():

param.requires\_grad\_(False) # 不可训练

#### fcn\_resnet50

语义分割网络

torchvision.models.segmentation.fcn\_resnet50()

#### fcn\_resnet101

语义分割网络

torchvision.models.segmentation.fcn\_resnet101()

#### deeplabv3\_resnet101

语义分割网络

torchvision.models.segmentation.deeplabv3\_resnet101()

#### deeplabv3\_resnet50

语义分割网络

torchvision.models.segmentation.deeplabv3\_resnet50()

#### fasterrcnn\_resnet50\_fpn

目标检测网络

在COCO2017数据集训练，有90个类别

具有ResNet-50-FPN结构的Fast R-CNN网络

torchvision.models.detection.fasterrcnn\_resnet50\_fpn()

#### maskrcnn\_resnet50\_fpn

目标检测网络

在COCO2017数据集训练，有90个类别

具有ResNet-50-FPN结构的Mask R-CNN网络

torchvision.models.detection.maskrcnn\_resnet50\_fpn()

#### keypointrcnn\_resnet50\_fpn

目标检测网络

在MSCOCO2017数据集训练，检测人体的关键点，17个类别，30万张

具有ResNet-50-FPN结构的Keypoint R-CNN网络

torchvision.models.detection.keypointrcnn\_resnet50\_fpn()

## pytorchtools库

可以引入早期停止技术

具体使用见<https://www.pythonf.cn/read/171508>

该文件不能使用cmd下载，

直接从下方的网站下载工具pytorchtools.py

<https://codechina.csdn.net/mirrors/bjarten/early-stopping-pytorch?utm_source=csdn_github_accelerator>

## 可视化库

### graphviz

from sklearn import tree

dot\_data = tree.export\_graphviz(model, out\_file=None,

feature\_names=data.feature\_names,

class\_names=data.target\_names,

filled=True)

graph = graphviz.Source(dot\_data, format="png")

graph.view()

### HiddenLayer

pip install HiddenLayer

pip install graphviz

dot -version # 表示成功

先下载Graphviz：<https://graphviz.org/download/>

添加路径C:\Graphviz\bin到用户变量

添加路径C:\Graphviz\bin\dot.exe到系统环境变量

重启

保存pytorch模型：

graph = hiddenlayer.build\_graph(model=model\_1,args=torch.zeros([516,13]))

graph.theme = hiddenlayer.graph.THEMES["blue"]

graph.save("graph.png",format="png")

保存tensorflow模型：需要tf1版本，会话和图机制

具体可以参考链接关于hiddenlayer的使用

<https://github.com/waleedka/hiddenlayer/tree/master/demos>

可视化训练过程：运行必须打开交互式后端，在pycharm的settings→tools→python scentific进行设置

canvas = hiddenlayer.Canvas() # 生成1个画布用于绘制

history = hiddenlayer.History() # 用于记录

# 在训练过程中记录的代码

history.log((epoch,step),train\_loss=train\_loss,test\_acc=acu,hidden\_weight=model\_1.regression.weight)

with canvas :

canvas.draw\_plot(history["train\_loss"])

canvas.draw\_plot(history["test\_acc"])

canvas.draw\_image(history["hidden\_weight"])

### torchviz

pip install torchviz

效果比HiddenLayer好

output = model(train\_xt)#需要计算输出,作为make\_dot的参数

graph\_=torchviz.make\_dot(output,params=dict(list(model.named\_parameters())+[('input',train\_xt)]))

graph\_.format = "png"

graph\_.directory = "graph"

graph\_.view()

### tensorboardX

pip install tensorboardX

例子：构建和保存网络的方式.py

import tensorboardX

①生成1个记录的类

summaryWriter = tensorboardX.SummaryWriter(log\_dir=path)

path为指定的合适的文件夹路径

②定义好全局的迭代步数

当前执行到的步数=当前epoch正在执行的step+以往执行完的epoch\*(max(step)+1)=epoch\*len(train\_dataset)

这里len(train\_dataset)实际上就是总的样本数N/批大小batch\_size

global\_step = step + 1 + epoch \* len(train\_dataset)

③指定每步记录的值，以及标签含义

summaryWriter.add\_scalar(tag=str,scalar\_value,global\_step)

④终端cd到合适的文件夹并输入命令

cd C:\Users\chenb\PycharmProjects\TensorflowStudy

tensorboard --logdir="Pytorch深度学习入门与实战\\tensorboardXlog"

⑤复制网址打开即可

#### 添加标量add\_scalar

add\_scalar(tag,scalar\_value,global\_step)

#### 添加直方图add\_histogram

add\_histogram(tag,scalar\_value,global\_step)

for name,param in model\_1.named\_parameters():

summaryWriter.add\_histogram(name,param.data.numpy(),global\_step)

#### 添加文本add\_text

add\_text(tag,text\_string,global\_step)

#### 添加网络结构add\_graph

add\_graph(model,input\_to\_model=None)

#### 添加音频add\_audio

add\_audio(tag,audio,iteration,sample\_rate)

#### 添加图像add\_image

add\_image(tag,x,iteration)

x的计算可通过语句batch\_x\_im = vutils.make\_grid(batch\_x,nrow=12)获得

### Visdom

pip install Visdom

要使用Visdom，就要先在终端开启监听命令：python -m visdom.server

打开网址：<http://localhost:8097>

然后再启动程序

vis = visdom.Visdom()

#### 图像image/images

# 在新环境中绘制单个图像

vis.image(im[0,...], win="image", env="new env")

# 绘制多个图像，需要指定nrow参数，即图像可视化排列的列数

vis.images(im[0,...], win="image", env="new env")

#### 文本text

vis.text("chenbei", win="text", env="new env")

#### 散点图scatter

vis.scatter(X=iris\_X[:,0:2],Y=iris\_y+1,win="scatter2d",env="main",opts=opts)

#### 线图line

vis.line(Y=ploty,X=plotx,win="line plot",env="main",opts=opts)

#### 茎叶图stem

vis.stem(Y=ploty,X=plotx,win="stem plot",env="main",opts=opts)

#### 热图heatmap

vis.heatmap(iris\_Corr,win="heatmap",env="main",opts=opts)

#### 其他类型图

音频audio

视频video

可视化matplotlib的图像matplotlib

条形图bar

直方图histogram

曲面图surf

盒型图boxplot

等高线图contour

箭头图quiver

网格图mesh

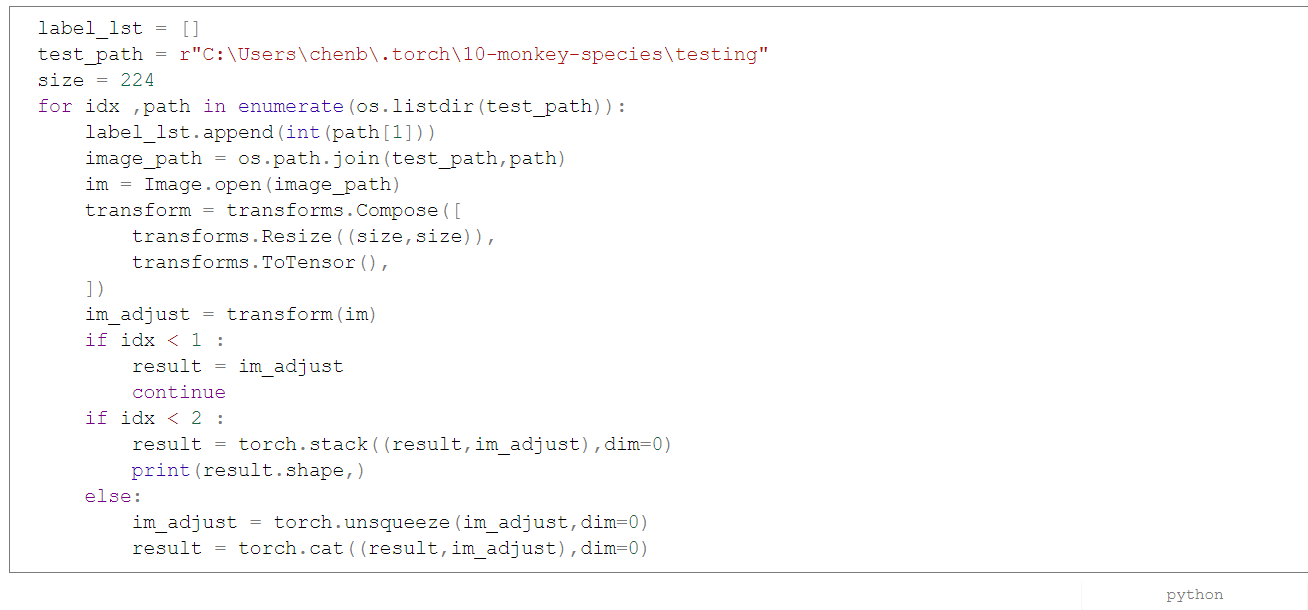
## Keras库

# 遇到的问题

## 如何使用得到四维张量作为测试数据集

思路：可以使用OS得到所有图片的路径，然后torchvision对uint8的数据进行处理，可以Resize调整形状，ToTensor转为0-1float32的的张量并调整颜色通道到第1个维度

第一次需要用一个temp等于im，第二次使用stack水平拼接得到（2，3，224，224）的形状，从第三次开始使用cat，就可以得到（N，3，224，224）的shape，可用于指定的输入网络数据



label\_lst = []

test\_path = r"C:\Users\chenb\.torch\10-monkey-species\testing"

size = 224

for idx ,path in enumerate(os.listdir(test\_path)):

label\_lst.append(int(path[1]))

image\_path = os.path.join(test\_path,path)

im = Image.open(image\_path)

transform = transforms.Compose([

transforms.Resize((size,size)),

transforms.ToTensor(),

])

im\_adjust = transform(im)

if idx < 1 :

result = im\_adjust

continue

if idx < 2 :

result = torch.stack((result,im\_adjust),dim=0)

print(result.shape,)

else:

im\_adjust = torch.unsqueeze(im\_adjust,dim=0)

result = torch.cat((result,im\_adjust),dim=0)