<https://zhuanlan.zhihu.com/p/87078890>得到数据文件以后，有一个问题就是里面得到的数据文件都是颠倒的，需要再做一个调整：

1. TestTest=cell(10000,2);
2. lll=[]
3. for i=1:10000

lll=Test{i,1}(:,:);

TestTest{i,1}(:,:)=lll.';

ll=Test{i,2};

TestTest{i,2}=ll;

end

1. 然后直接点击TestTest进行保存
2. 对Train重复

然后将前面得到的数据的每个cell中的矩阵都拉成列向量：

1. TestTest1=cell(10000,2);
2. for i=1:10000

lll=TestTest{i,1}(:,:);

lll=reshape(lll,[Nx\*Ny,1]);

TestTest1{i,1}(:,:)=lll;

ll=TestTest{i,2};

TestTest1{i,2}=ll;

end

1. 保存
2. 对TrainTrain重复

运行data.m文件进行数据生成

关于数据为什么要归一化：

<https://www.cnblogs.com/bjwu/p/8977141.html>

现在原始图片已经有了，散斑数据集也已经产生了，包括jpg和mat文件都有，可以通过几种方式进行，要么通过mat转npy要么把jpg转成npy

Test是原始的还原的数据文件

TestTest是进行了九十度旋转的数据文件

TestTest1是每个图片的像素都被拉成一列以后的数据文件，用来生成图片

TestTest2是TestTest前两千个文件，是用来进行训练的数据，这就是我们要使用的数据，但是要注意这里是散射前的原始数据，这里我们不再需要第二列，可以直接删除，这里注意下面生成测试数据的时候也要注意这点

TestTest3是TestTest两千到两千一百的数据文件，是用来进行测试的数据

Test7是测试数据集的第一个数据

然后现在使用的生成散斑数据的方法是先生成它的mat文件然后再把它变成npy，这里就用TestTest5生成然后保存，然后先预处理直到格式对了以后再用来训练

TestTest4是错误的，我不小心把它搞成列向量了，要搞就搞成一样的存储格式，所以有了TestTest5

然后现在需要的是128\*128的图片格式，我需要创建TestTest6把原始数据存储然后送入程序运行并且把结果保存在同为128\*128的TestTest8里面，TestTest7用来保存6拉成的长条然后用来生成图片

for i=1:2000

lll=TestTest{i,1}(:,:);

lll = imresize(lll, [128 128]);

TestTest6{i,1}(:,:)=lll;

end

>> TestTest6=cell(2000,1);

>> for i=1:2000

lll=TestTest{i,1}(:,:);

lll = imresize(lll, [32 32]);

TestTest6{i,1}(:,:)=lll;

end

>> TestTest7=cell(2000,1);

>> for i=1:2000

lll=TestTest6{i,1}(:,:);

lll=reshape(lll,[32\*32,1]);

TestTest7{i,1}(:,:)=lll;

end

>> TestTest8=cell(2000,1);

运行了如上代码以后需要将67保存了以后再运行data.m

运行结束以后记得保存8即可

但是发现liyunzhe的论文的代码里面输出也就是标签是包含两种的，一种黑底白字一种白底黑字，所以创建一个TestTest9来存放两列的数据

TestTest9=cell(2000,1);

加一行lll = imcomplement(lll)然后把列换为2即可

问题：plt.imshow()无法显示图像

解决方法:添加：plt.show()，即

plt.imshow(image)  #image表示待处理的图像

plt.show()

原理：plt.imshow()函数负责对图像进行处理，并显示其格式，而plt.show()则是将plt.imshow()处理后的函数显示出来。

1.figure语法及操作

(1)figure语法说明

figure(num=None, figsize=None, dpi=None, facecolor=None, edgecolor=None, frameon=True)

num:图像编号或名称，数字为编号 ，字符串为名称

figsize:指定figure的宽和高，单位为英寸；

dpi参数指定绘图对象的分辨率，即每英寸多少个像素，缺省值为80      1英寸等于2.5cm,A4纸是 21\*30cm的纸张

facecolor:背景颜色

edgecolor:边框颜色

frameon:是否显示边框

(2)例子:

import matplotlib.pyplot as plt

创建自定义图像

fig=plt.figure(figsize=(4,3),facecolor='blue')

plt.show()

2.subplot创建单个子图

(1) subplot语法

subplot(nrows,ncols,sharex,sharey,subplot\_kw,\*\*fig\_kw)

subplot可以规划figure划分为n个子图，但每条subplot命令只会创建一个子图 ，参考下面例子。

(2)例子

import numpy as np

import matplotlib.pyplot as plt

x = np.arange(0, 100)

#作图1

plt.subplot(221)

plt.plot(x, x)

#作图2

plt.subplot(222)

plt.plot(x, -x)

 #作图3

plt.subplot(223)

plt.plot(x, x \*\* 2)

plt.grid(color='r', linestyle='--', linewidth=1,alpha=0.3)

#作图4

plt.subplot(224)

plt.plot(x, np.log(x))

plt.show()

3.subplots创建多个子图

(1)subplots语法

subplots参数与subplots相似

(2)例子

import numpy as np

import matplotlib.pyplot as plt

x = np.arange(0, 100)

#划分子图

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

ax1=axes[0,0]

ax2=axes[0,1]

ax3=axes[1,0]

ax4=axes[1,1]

#作图1

ax1.plot(x, x)

#作图2

ax2.plot(x, -x)

 #作图3

ax3.plot(x, x \*\* 2)

ax3.grid(color='r', linestyle='--', linewidth=1,alpha=0.3)

#作图4

ax4.plot(x, np.log(x))

plt.show()

4.面向对象API：add\_subplots与add\_axes新增子图或区域

add\_subplot与add\_axes都是面对象figure编程的，pyplot api中没有此命令

(1)add\_subplot新增子图

add\_subplot的参数与subplots的相似

import numpy as np

import matplotlib.pyplot as plt

x = np.arange(0, 100)

#新建figure对象

fig=plt.figure()

#新建子图1

ax1=fig.add\_subplot(2,2,1)

ax1.plot(x, x)

#新建子图3

ax3=fig.add\_subplot(2,2,3)

ax3.plot(x, x \*\* 2)

ax3.grid(color='r', linestyle='--', linewidth=1,alpha=0.3)

#新建子图4

ax4=fig.add\_subplot(2,2,4)

ax4.plot(x, np.log(x))

plt.show()

可以用来做一些子图。。。图中图。。。

(2)add\_axes新增子区域

add\_axes为新增子区域，该区域可以座落在figure内任意位置，且该区域可任意设置大小

add\_axes参数可参考官方文档:http://matplotlib.org/api/\_as\_gen/matplotlib.figure.Figure.html#matplotlib.figure.Figure

import numpy as np

import matplotlib.pyplot as plt

#新建figure

fig = plt.figure()

# 定义数据

x = [1, 2, 3, 4, 5, 6, 7]

y = [1, 3, 4, 2, 5, 8, 6]

#新建区域ax1

#figure的百分比,从figure 10%的位置开始绘制, 宽高是figure的80%

left, bottom, width, height = 0.1, 0.1, 0.8, 0.8

# 获得绘制的句柄

ax1 = fig.add\_axes([left, bottom, width, height])

ax1.plot(x, y, 'r')

ax1.set\_title('area1')

#新增区域ax2,嵌套在ax1内

left, bottom, width, height = 0.2, 0.6, 0.25, 0.25

# 获得绘制的句柄

ax2 = fig.add\_axes([left, bottom, width, height])

ax2.plot(x,y, 'b')

ax2.set\_title('area2')

plt.show()

**z-score 标准化(zero-mean normalization)**

最常见的标准化方法就是Z标准化，也是SPSS中最为常用的标准化方法，spss默认的标准化方法就是z-score标准化。

也叫标准差标准化，这种方法给予原始数据的均值（mean）和标准差（standard deviation）进行数据的标准化。

经过处理的数据符合标准正态分布，即均值为0，标准差为1，其转化函数为：

*x*∗=*x*−*μσ*

其中μ为所有样本数据的均值，σ为所有样本数据的标准差。

z-score标准化方法适用于属性A的最大值和最小值未知的情况，或有超出取值范围的离群数据的情况。

**标准化的公式很简单，步骤如下**

　　1.求出各变量（指标）的算术平均值（数学期望）xi和标准差si ；  
　　2.进行标准化处理：  
　　zij=（xij－xi）/si  
　　其中：zij为标准化后的变量值；xij为实际变量值。  
　　3.将逆指标前的正负号对调。  
　　标准化后的变量值围绕0上下波动，大于0说明高于平均水平，小于0说明低于平均水平。

def z\_score(x, axis):

x = np.array(x).astype(float)

xr = np.rollaxis(x, axis=axis)

xr -= np.mean(x, axis=axis)

xr /= np.std(x, axis=axis)

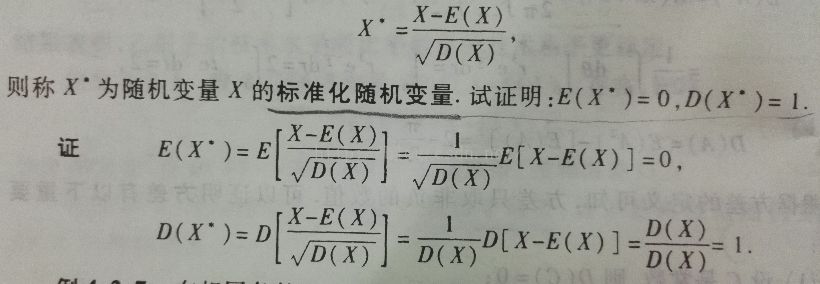
# print(x)

return x

**为什么z-score 标准化后的数据标准差为1?**

x-μ只改变均值，标准差不变，所以均值变为0

(x-μ)/σ只会使标准差除以σ倍，所以标准差变为1



发现原先代码中包含些许错误，修改的时候命名顺序如下：

Ko中第一列存储散斑图片，第二列存储散斑图片的黑底白图

Ko1中存储前两列图片拉长的图片