模式识别与机器学习大作业 细胞图像识别与分割

门恺文 自 83 2018011501

摘 要：本次大作业面临的问题是对未标注的细胞图像运用机器学习的方法进行分割标注。其中 dataset1 对应着第一个任务，训练集标注比较充分，且细胞相对较小、较稀疏，可以使用传统方法或深度学习方法；dataset2 对应着第二个任务，训练集的标注较少，且细胞大而紧密，不容易分开，使用深度学习的方法会得到较好的结果。

代码在 tensorflow.keras 框架下编写和运行，采用语义分割加后处理的方法。对于第一个训练集，搭建 Unet 卷积神经网络，对原训练图像扭曲、平移得到更多的训练样本，将标签二值化，之后用训练样本对搭建好的 Unet 网络进行训练，得到训练模型。采用此训练模型对测试集进行预测，得到预测结果。之后对预测结果二值化，并采用 DFS 给每个细胞不同的标签，即可得到多分类结果。最终结果 jaccard 相似度为 0.640389。对于第二个训练集，采取相似的方法，并进一步调整模型参数，但结果不尽如人意，尝试了 Unet 论文用到的训练集，结果仍然不好，所以最终采取了 dataset1_demo 中的方法进行传统的分割。jaccard 相似度为 0.06 左右。

关键词：细胞分割； 深度学习； 卷积神经网络

1. 文献调研

对于细胞分割问题，既可以使用传统的方法，也可以使用深度学习的方法。传统的方法主要包括基于阈值的分割、基于边缘的分割、基于区域的分割、基于图论的分割、基于能量泛函的分割[1]、基于聚类的分割[2]。深度学习方法主要包括滑窗法、Unet 等[3]。

传统方法中，基于阈值的分割包括直方图双峰法、固定阈值分割、半阈值分割、迭代阈值图像分割和自适应阈值图像分割[4]。直方图双峰法是典型的单域值分割方法，该方法依托于图像中有明显的背景前景，寻找灰度图双峰之间的谷值作为阈值，而本次作业中细胞图像前景与背景不是很分明，所以这种方法不适合。固定阈值分割是选定一个阈值，大于该阈值的设为一个标签，小于该阈值的设为另一种标签。半阈值分割是一种增强边缘的算法，把图象分割为 N 等份，计算每一份中前景点的数量，当前景点达到一定数量时，进行阈值分割，否则不做处理，通过此方法可以使图像边沿分明[5]。

迭代阈值图像分割是采用阈值逼近的方法进行阈值选取。自适应阈值图像分割应用于图像背景亮度不同时的自适应调整[4]。

基于边缘的分割方法是通过图像不同区域边界线上的像素值突变实现的。一般的，像素值的突变包括阶跃型和屋顶型，边缘角点和兴趣点的检测器包括 Canny 边缘检测器、Harris 角点检测器、SIFT 检测器、SURF 检测器。这个在此次大作业中可以用于将预测出图像的边缘分开。

基于区域的分割是按照图像的相似性准则划分为不同的区域，包括种子区域生长法、区域分裂合并法、分水岭法。种子区域生长法根据像素的相似性聚集像素点来达到区域生长。区域分裂合并法是将图像分为若干互不相交的区域，之后按照一定的规则分裂合并。分水岭法对微弱的边缘有着良好的响应，但图像中的噪声会使分水岭算法过分割。[1]

基于图论的分割方法采用了图割的方法，首先根据一个图的前景和背景训练贝叶斯分类器，之后根据节点建立起一个图，并以预测出的两点间的权重添加边，之后对图分割。[2]

基于能量泛函的分割主要是活动轮廓模型及发展起来的算法，而活动模型分为参数活动轮廓模型和几何活动轮廓模型。这种方法通过求解能量泛函的最小值找到目标的轮廓，实现图像分割。

基于聚类的分割是采用 KMeans 等聚类算法，将图像中像素一致的相近的点聚为一类，之后通过二值化可以达到分割的目的。[2]

传统算法在细胞比较稀疏的情况下可以得到较好的分割算法，但在细胞密集的情况下较难分割出细胞。

深度学习领域，滑窗法预测窗重心有着很好的分割效果，可以在训练样本不多的情况下取得精确的、像素级别的分类。然而，这种方法有两个缺点：一是十分慢且有冗余，二是输入尺度和性能之间的矛盾，即感受野和分割精度呈负相关。而基于 FCN 网络的 Unet 则可以通过很少的训练样本取得很好的分割效果，采用 U 形网络，在对称层之间也有连接，充分利用已有的图像，达到很好的分割效果。[6]所以本次作业最终选择了 Unet。

2. 数据处理流程

2.1 dataset1

2.1.1 训练集

由于需要扩充训练集，所以对训练集进行数据增强，即通过 ImageDataGenerator 所带的

flow_from_directory 函数，调整函数的参数，对训练集图像进行平移、旋转、翻转等操作，并合并为一个生成器。为了统一训练样本，并为语义分割做准备，将图像归一化，并把标签二值化。由于本次训练的标签是背景为零，细胞标注有不同的数字，所以二值化的实现设计为原本为 0 的保持，原本大于零的设为 1。同时，为了加快训练速度，预处理阶段将训练集图片大小调整为（256，256）。

2.1.1 测试集

测试集数据的处理主要是测试集的读取和调整。测试集本身就是灰度图，所以读取时采用 cv2.imread 读入原图。在预测过程中发现了预测样本标签和测试集标签对不齐的情况，所以最后采用了按标签顺序读入，而不是直接按文件夹中的文件顺序读入。训练集生成的增强数据由于多了一维 batchsize，所以训练集数据大小是 2*256*256*1，为了与训练集统一，测试集数据也要相应调整到这个大小，即将测试集数据大小调整为（256，256），并归一化，之后在每个样本前后各增加一列。

2.2 dataset2

dataset2 最终的实现是采用了传统的分割算法，只涉及测试集的数据预处理。需要对每个图像标准化，即转化成 uint8 格式，并统一为三通道彩色图像。

3. 算法原理

算法包括模型搭建、数据预处理、模型训练和预测、数据后处理四部分。

3.1 dataset1

3.1.1 模型搭建

采用 unet 结构，输入为预加载权重和输入数据的大小，输出为 Unet 模型。包含十层卷积神经网络，最后一层是输出层，是使用 sigmoid 激活函数的卷积层，实现二分类；第五层是中间的连接层，采用两次卷积，使用 relu 激活函数，边缘处设为不补零，并添加 dropout 层避免过拟合；前四层每层均有两次卷积和一次池化，卷积层采用 relu 激活函数，池化层池化大小均为 (2,2)；除输出层外的后四层与前四层一一对应，对上一层进行上采样，并卷积，之后建立与对称层的联系，从而对应到对称的层，之后再经过两次卷积，卷积层的激活函数均设为 relu，边缘处均设为不补零。

optimizer 采用 Adam，输入为学习率 lr，和动量参数 beta_1, beta_2, epsilon，输出为该优化器。设置参照 keras 中文文档[7]和自己实际训练模型

时得到的训练结果。

对于 loss 函数，由于 dataset1 样本背景较多，前景较小，所以需要调整训练时损失函数中前景所占比重，所以 loss 函数采用 focal-loss[8]。输入为前景背景损失代价的指数 gamma 和平衡参数 alpha，输出为 focal_loss_fixed 损失函数。

本次作业中一共使用了三个评价函数，一个是常用的 accuracy，另外添加了与 focal-loss 相对应的 dice_coef。同时，由于 jaccard 相似度与 IoU 类似，所以加入了 keras 自带的 MeanIoU。

3.1.2 数据预处理

数据预处理包含了三个函数，train_Generator，adjustData，test_Generator。前两个是对训练集数据的预处理，后一个是对测试集数据的预处理。

3.1.2.1 train_Generator

用于生成训练集，基本原理是借助数据增强函数 ImageDataGenerator.flow_from_directory 读取训练集并进行数据增强。输入为：

batch_size：生成增强数据后每次读入的大小；

train_path：原训练集图像路径；

mask_path：原训练集标签路径；

image_folder：图像文件夹名称；

mask_folder：标签文件夹名称；

aug_dict：增强参数，包括旋转角度、平移距离等；

image_color_mode：生成的图像颜色模式，默认为灰度；

mask_color_mode：生成的标签颜色模式，默认为灰度；

image_save_prefix：增强后的图像保存前缀；

mask_save_prefix：增强后的标签保存前缀；

flag_multi_class：是否是多分类，默认为 False；

num_class：分类数，默认为 2；

save_to_dir：增强后的图片保存路径，默认为 None；

target_size：增强后图片的尺寸，默认为(256, 256)；

seed：数据增强时设置的随机种子，默认为 1。

函数输出为训练图像及标签的生成器。

算法变量 image_datagen 和 mask_datagen 分别为图像和标签的 ImageDataGenerator，用于增强数据；image_generator 和 mask_generator 分别用于承接图像和标签生成的数据；

train_generator 用来将增强后的图片和标签打包。img, mask 是两个循环变量，用于循环调整每组图片和标签。算法的具体执行步骤为先生成两个数据增强器，之后通过数据增强器生成增强数据，之后将增强的数据分别循环调整。

3.1.2.2 adjustData

训练集循环调整时用到的函数，在二分类时用于把图像归一化，标签二值化。

输入：

img：要调整的图像；

mask：要调整的标签；

flag_multi_class：是否进行多分类的标志，此处一致为 False；

num_class：分成几类，此处一致为 2.

输出：调整后的图像和标签：img，mask。

该函数中没有其他变量，执行步骤为先对 img,mask 标准化为 uint8 格式，之后将 img/255 归一化，把 mask/255 后，mask 中为 0 的像素点继续保持，大于 0 的像素点设为 1.

3.1.2.3 test_Generator

读取测试数据，并调整为适合模型的大小。

输入：

test_path：测试集路径；

target_size：读入后的调整大小，默认为(256,256)；

flag_multi_class：是否是多分类，默认为 False；

n：测试数据数量，默认为 33.

输出：打包好的测试集。

算法变量 i 为循环变量，用于循环处理文件夹中的图片；img 为每次读入的图片。具体执行步骤为以 i 为循环，依次读入对应的文件名，将 img/255 归一化，调整大小为（256，256），在 img 前后各加一列。

3.1.3 模型训练与预测

用预处理后的训练数据训练模型，再用模型预测测试数据。输入为搭建好的模型，预处理后的训练集和测试集，输出为测试集预测结果。

算法变量 data_gen_args 是设置增强时的模型参数，myGene 是训练集生成器，model_checkpoint 用于在过拟合之前及时保存模型，testGene 是测试集生成器，results 是测试结果。算法具体执行步骤为先得到训练集生成器，然后进行训练，之后用训练好的模型测试，得到测试结果。

3.1.4 数据后处理

主要是调整图像大小、二值化、标准化并给每个细胞标上不同的标签。

整体输入为预测结果，输出为后处理后的图像。其中子函数 DFS 的输入为数组 result，两个数组下标 ij，颜色 color，无输出。子函数 IndexInRange 输入为下标 i,边界 bond，默认为 628。输出为 i 是否满足 0<= i <bond 的布尔变量。

算法中间变量为：循环变量 k 遍历每个预测结果，循环变量 i，j 遍历结果的每个像素；大小为 256 的布尔型数组 color_used 用于表示 256 种颜色的使用情况，避免重复使用；临时变量 result 是 results 的第 k 个元素，方便处理；color 是每次遇到没有更新标签的细胞时生成的 0-255 之间的随机数，该随机数跟着 DFS 扩展到整个细胞区域，使得每个细胞有不同的标签。

具体执行步骤为：通过一个循环，遍历每个预测结果，每次循环生成一个新的 color_used 数组,除了 0 和 255 外其他位置的初始值为 False。result 用于承接循环到的 results 成员，将 result 大小调整回（628，628），并以 0.5 为阈值对 result 二值化,小于 0.5 的设为 0，大于 0.5 的设为 1。然后将 result 乘 255，并标准化为 uint8。之后遍历 result 的每个像素，发现像素值为 255 时，生成一个 0-255 的随机数 color,用 color_used 检测该 color 是否被用过，如果用过了，则生成新的 color，直到是没用过的。之后把 color_uesd 的该位置设为 True，然后 color 进入递归函数 DFS，DFS 采用深度优先搜索给该像素点及周围的所有值为 255 的像素点上色。所有细胞上色完成后存入文件夹。实际运行过程中，出现过由于直接执行所有结果的循环程序崩溃的情况，但是分开来执行每个结果的处理和存储不会使程序崩溃。即每次手动改变 k 的值，该段程序附在循环处理的后面。

3.2 dataset2

由于 dataset2 没有探索出好的深度学习的办法，所以使用的是 demo 中给出的传统方法。

3.2.1 unit16b2uint8

输入为待调整图像 img，输出为 uint8 格式的 img。原理为将数据格式设为 uint8.具体执行步骤为输入 img 判断是否是 uint8 格式，若是，则直接返回，若不是，则返回类型转换之后的数据。

3.2.2 img_standardization

输入为原图像 img，输出为调整后 img。将所有 img 转化为 uint8 格式和三通道彩色图。具体执行步骤为先将 img 转换为 uint8 格式，然后判断 img

维度，如果是二维，则扩展为三维，第三维长度为 3；如果是三维，则直接返回；否则输出维数报错。

3.2.3 load_images

输入为文件夹的路径，输出为 images 数组，用于将数据读入。依次读入文件夹中的每个文件，进行标准化，之后合并成一个列表。

3.2.4 bgr_to_gray

输入为待转换图像，输出为转换后图像。把彩色图转化为灰度图。

3.2.5 BinaryThresholding

分割算法所在类，输入为阈值 Threshold，图像 img，输出为分割后的结果 label_img。具体执行步骤为先得到 img 的灰度图 gray，然后对 gray 进行阈值分割，得到掩模，然后对掩模中值滤波除噪，之后进行轮廓提取，返回分割好的图像。

3.2.6 主函数

主函数用于调用分割的类并生成分割结果存储起来。具体执行步骤为以 110 为阈值，调用 BinaryThresholding，建立 result_path，然后调用 load_images 将测试集数据读入，之后遍历每个样本，进行分割和存储。

4 实现过程

4.1 dataset1

整体架构（主函数）如图 1

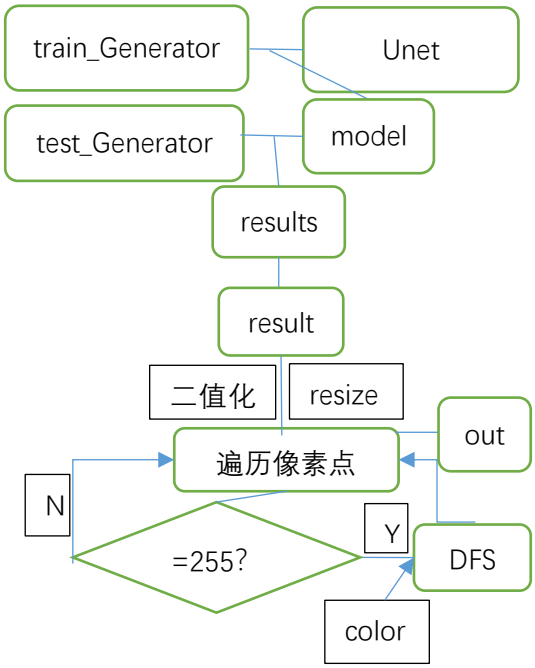

图 1

4.1.1 Unet

Unet 的整体结构如图 2.

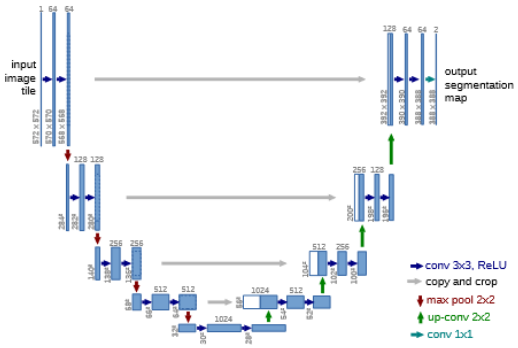

图 2

有四层下连接层，四层上连接层，一层中间层，一层输出层，总体呈 U 形。每个下连接层都有两次卷积，一次池化。卷积层用 Conv2D，输出节点从 64 开始，每次增加两倍，activation = 'relu'，padding = 'same'，kernel_initializer = 'he_normal'；

池化层用 MaxPooling2D，pool_size=(2, 2)。上连接层有一次上采样，三次卷积，和与对称层的连接。第一次卷积的输入是上一层的上采样 UpSampling2D(size = (2,2))，与对称层的连接采用 concatenate，其中 axis=3，卷积的参数与下连接层一致。中间层采用两次卷积，一个避免过拟合的 Dropout，卷积层的设置与上连接层一致，Dropout 层参数设为 0.5。

4.1.2 adjustData

功能如图 3.相当于图像和标签的统一化。

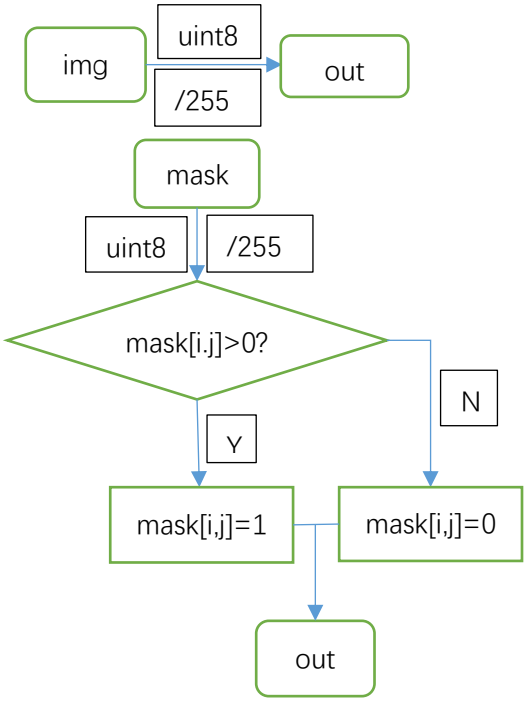

图 3

4.1.3 train_Generator

功能如图 4，使用 ImageDataGenerator 的 flow_from_directory 函数进行数据增强。

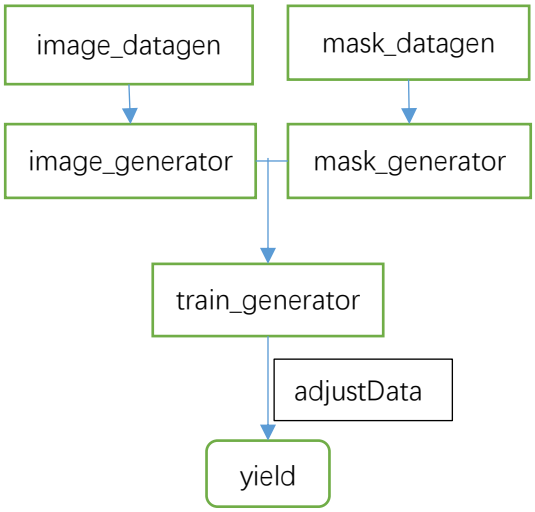

图 4

4.1.4 test_Generator

功能如图 5，生成测试集。

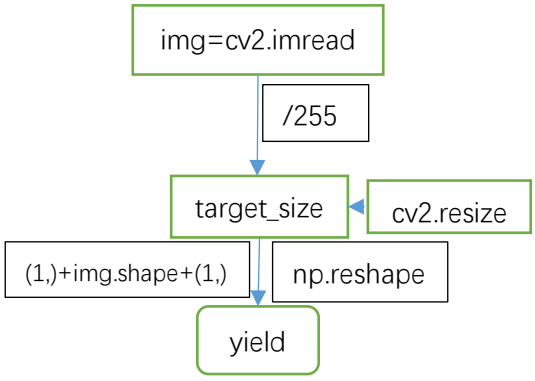

图 5

4.1.5 IndexInRange & DFS

IndexInRange 仅在 DFS 中使用过，所以一起介绍。IndexInRange 返回下标是否在范围内，DFS 是一个递归函数，实现过程如图 6.

```
def DFS(result, i, j, color):
    result[i, j] = color
    if (IndexInRange(i - 1) and result[i - 1, j] == 255):
        DFS(result, i - 1, j, color)
    if (IndexInRange(i + 1) and result[i + 1, j] == 255):
        DFS(result, i + 1, j, color)
    if (IndexInRange(j - 1) and result[i, j - 1] == 255):
        DFS(result, i, j - 1, color)
    if (IndexInRange(j + 1) and result[i, j + 1] == 255):
        DFS(result, i, j + 1, color)
    return
```

图 6

5 实验结果与模型性能分析

5.1 dataset1

对于 Unet 模型，尝试了不同的 optimizer，loss，metrics。optimizer 中的 Adam 设置有 Ir 分别为 1e-4 和 1e-5，以及加入其他参数调试，得到的 accuracy 如表 1.

optimizer 参数	accuracy
Ir=1e-4	0.88
Ir=1e-5	0.97(不稳定)
Ir=1e-5, beta_1=0.9, beta_2=0.999, epsilon=1e-08	0.97(较稳定)

表 1

可见当 Ir 为 1e-4 时，accuracy 很低，这说明学习率太高了，需要调低；当 Ir 为 1e-5 时，虽然 accuracy 显著提高，但是不稳定，仍然会有很低的情况，说明参数设置不完善；当同时设置了 beta_1，beta_2，epsilon，accuracy 变得较高且稳定。所以对于 optimizer 参数最后选择了第三种。

loss 尝试过 mse，focal_loss，dice_coef_loss，得到预测结果观测如表 2.

loss 函数	预测结果
mse	有大量背景被预测成了前景
focal_loss	比较适合背景较多前景较少的情况
dice_coef_loss	分割效果不够好

表 2

可见，在本次大作业中，由于背景和前景的比例不均衡，所以使用 mse 时会出现预测结果不够好的问题，而 focal_loss 对这类问题解决得比较好，所以 loss 函数最终选择了 focal_loss。

对于评价函数，影响如表 3.

评价函数	accuracy
只有 accuracy	较低
加入 MeanIoU	显著提高
加入 dice_coef	变化不明显

表 3

可见，当三个评价函数均加入时，结果最好且最稳定，原因可能是对于细胞分割的问题，MeanIoU 和 dice_coef 可以较准确反映出模型的性能，使得模型相应做出调整，准确率较高。所以评价函数三者均加入。

5.2 dataset2

dataset2 有过 Unet 网络的尝试，和传统算法的尝试。

Unet 网络尝试如表 4.

尝试	效果
loss 函数采用 mse	分割结果较差
采用网上的数据集	分割结果较差
采用原本的已标记的数据集并增强	分割效果较差
采用 dataset1 的数据集	分割效果较差

表 4

由表 4 可知，已有的尝试得到的分割效果都比较差，主要原因是没有足够多的合适的数据集，网络参数设置不够合理，以及最后时间紧张但运行太慢，只好退而求其次，采用传统方法。

6 总结

Unet 网络对于医学影像的分割有着优良的性能，本次大作业依托 Unet 完成了细胞分割，采用了数据增强等数据预处理方法和深度优先搜索等数据后处理方法。并在实验和资料查询过程中调整了参数，得到了较好的结果。

7 感想

本次大作业是我第一次接触到神经网络，有很多陌生的东西。在开始的时候会遇到不明白模型的输入输出有什么具体的要求，不知道数据增强是什么原理，也不知道应该在何处改进等问题，但是在第一次终于跑出来了一个不是全零的结果时那由衷的喜悦却让我记忆深刻。虽然这结果不太好，但是还是感到本次大作业对我有很大的帮助，让我对神经网络有了初步的认识。也期望从排名靠前的同学那里学习到更好的方法。

8 参考文献

[1] Nango 明楠. 图像分割 传统方法整理 [EB/OL]. Medium. https://zhuanlan.zhihu.com/p/30732385

[2] venus. python 计算机视觉学习——图像分割[EB/OL]. Medium. [2019-07-16]. https://blog.csdn.net/qimingxia/article/details/95723482

[3] MaggieQuan. 深度学习之图像分割[EB/OL]. Medium. [2019-05-06]. https://blog.csdn.net/qq_36492210/article/details/89875708

[4] wangduo. 基于阈值的图像分割方法[EB/OL]. Medium. [2016-10-31]. https://www.cnblogs.com/wangduo/

p/5556903.html

[5] 潘振兴，谢涛. 基于半阈值的字符分割与识别研究[J]. 国防科学技术大学研究生院. P22

[6] lavendelion. 经典论文解析——Unet 和 Vnet——图像分割[EB/OL]. Medium. [2019-11-17]. https://blog.csdn.net/weixin_41424926/article/details/103105801

[7] keras 中文文档. 优化器的用法[EB/OL]. Medium. https://keras.io/zh/optimizers/

[8] 独孤金泽. focal-loss 的 keras 实现[EB/OL]. Medium. [2019-05-22]. https://blog.csdn.net/qq_35747066/article/details/90437002