**电力巡检智能缺陷检测**

**摘　要**

架空输电线路中的绝缘子易产生自爆缺失故障，根据大量高分辨率航拍的绝缘子图像，本文设计了一套基于深度学习与主成分分析的绝缘子自爆故障检测的智能算法。

**第1步，对数据进行预处理。**我们对所给图像分别进行放大、缩小、旋转、改变色彩和饱和度等变换，以扩充得到了19倍的图像集，并以固定的子块大小对每张图片进行切割分块，得到4万多个子块作为训练样本。

**第2步，对绝缘子区域进行语义分割**。我们在金字塔场景解析网络**PSPNet**用20K张图片预训练的模型的基础上，我们用自己的子块图像及其二值掩模作为输入在一台具有TitianXp GPU的机器上进行训练。因为绝缘子的区域面积相对整幅图像比较小，我们还**提出来了一种新损失函数，对绝缘子区域的损失进行了加权，能更完整地提取绝缘子区域**。我们的模型对训练样本能达到99%的分类精度，对未知的测试样本能达到98%的分类精度，对绝缘子区域的分类精度更高。原图的标准掩模图像与其相应预测结果掩模图像的Dice系数的平均值为**0.8474。**

**第3步，规范化绝缘子区域。**提取了绝缘子所在区域的二值掩模之后，我们找出了每个连通区域，并**提出了一种基于主成分的旋转和裁剪算法**，把每串绝缘子单独切分成一个独立的图像，去掉一些噪音，并且旋转到水平方向，规范化为相同的尺寸。

**第4步，对自爆点进行定位**。把每串绝缘子的规范化图像及其自爆点区域作为**YOLO网络模型**的输入进行训练，由于图像样本非常单一，**我们利用K-means聚类设计了并指定了几种锚框**，网络模型就能够以非常高的精度定位自爆点的位置，其IOU的平均值为**0.8749**。

本文设计的基于深度学习和主成分分析的算法能够在不同光照条件、不同拍摄角度以及复杂背景干扰下实现绝缘子串的识别与分割，且处理时间短、精度高，适用于影像分辨率高且背景复杂的绝缘子自爆缺陷自动检测。

**关键词：**绝缘子分割；自爆缺陷定位；语义分割；目标检测；深度学习

**1 引言**

绝缘子自爆是高压输电线路中常见的缺陷，同时也为了输电线路的安全可靠，电网部门需要定期对输电变电系统进行巡逻，勘测是否有自爆绝缘子的存在。传统的巡查做法是通过人工进行检测，但是这样会存在劳动强度大，工作效率低，工人人身安全无法得到保障等问题。

在国内外，图像处理技术和机器视觉广泛应用于电力巡检中的识别和检测任务。文献[1]使用结合LAB彩色空间 、“最大类间方差法”和 “面积形态学”，建立数学模型。文献［2］通过搭建卷积神经网络，在由5个卷积池化模块和2 个全连接模块组成的经典架构的基础上，抽取绝缘子的特征融入自组织特征映射网络中实现检测。文献［3］使用Faster R-CNN［7］模 型，结合Resnet-101深度残差卷积神经网络，对所有感兴趣区域进行分类，再对Bounding Box 回归和坐标进行修正实现了绝缘子串的识别和定位，但得到的结果还是不够精细，比较模糊和平滑，对图像中的细节不敏感。文献［4］提出了一种基于深度学习U-Net［8］网络的航拍绝缘子串分割方法，进行像素提取和定位，提高了定位精度，目前它主要应用于样本较小的医学影像的处理，在其他领域的应用较少。文献［1－2］均采用较为经典的方法，识别速度较慢，精度较低。

国内外学者也开展了通用图像的分割方法的研究。文献［3］以 FCN［9］网络为基础，通过微调使原网络适应新的绝缘子串数据集，实现了在复杂背景下 绝缘子串的语义分割，该方法对各个像素进行分类，没有充分考虑像素与像素之间的关系。忽略了在通常的基于像素分类的分割方法中使用的空间规整（spatial regularization）步骤，缺乏空间一致性。文献［5］提出了一种基于三维OTSU 的图像分割方法，该方法计算复杂度较高，分割效率低。文献[10-13]的网络模型也用于语义分割与目标检测，其算法适用于相关领域，如医学，地质学等。

本文以绝缘子巡视中的绝缘子自爆这一故障为目标，主要实现绝缘子串珠分割和绝缘子自爆识别和定位两部分内容。由于无人机图片较大一般为（4096\*2160），绝缘子串珠占据图片中很小的一部分区域， 需要设计图像分割算法，对绝缘子串珠所在的区域进行分割。根据分割图像初步识别绝缘子所在的位置，并对绝缘子串珠进行分割，而后根据所给出的标记样本的 Ground Truth 构建自爆绝缘子识别模型。

本文的计算平台依托后台图形计算服务器，其配置是Window10 系统、Intel 酷睿7代CPU（主频3. 2 GHz、6 核心、12 线程）、64 G 内存、Nvidia Geforce GTX Titan Xp 图形处理器（12 G 显存）。深度学习框架为基于Python的Tensorflow 和Keras。

本文以深度学习算法为依托，采用速度快、效率高的PSPNet［14］和YOLO［15］的网络模型。提出来了一种新损失函数对PSPNet进行训练，对绝缘子区域的损失进行了加权，能更完整地提取绝缘子区域。还提出了一种基于主成分的旋转和裁剪算法，把每串绝缘子单独切分成一个独立的图像。实验表明，该方法能够在不同光照条件、不同拍摄角度以及复杂背景噪声干扰下的航拍图像中实现绝缘子串的识别与分割，且处理时间短、精度高、鲁棒性强。

**2 数据准备与预处理**

题目给出了40张高分辨率的航拍绝缘子的图像及其掩模，如图1是其中一张分辨率为的原图像及其掩模。



图1 航拍高分辨率的绝缘子图像及其掩模

绝缘子有多种颜色，在航拍图像中，其颜色也会受到地表植被、泛绿的湖水的影响。为了获得更加丰富的训练样本，使得PSPNet语义分割模型训练效果更好，得到分割出绝缘子串珠的精度更高，我们将对原图像进行增强，具体Python代码的实现可见附录一。使用到的图像变换有：

1. 图像缩放。分别对原图进行0.6、0.8、1.2和1.4倍进行缩放。
2. 图像翻转。分别对原图进行左右和镜像进行翻转。
3. 图像旋转。分别对原图旋转20、40、60、90、-20、-40、-60度。图2是一个原图旋转得到的图像。

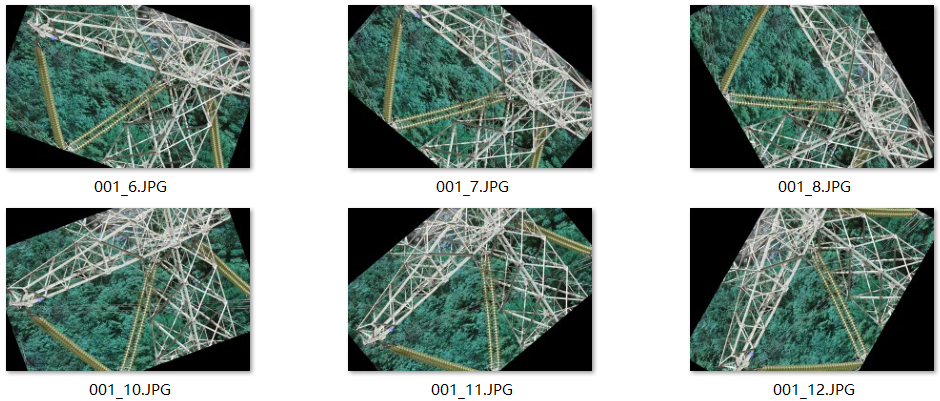


图2 原图旋转后得到的图像

1. 色彩变换。分别对原图进行HSV变换，然后使得饱和度增加10、20、-10、-20，然后再变换回RGB图像。并分别把原图转换为灰度图。图3是色彩变换后得到的图像。

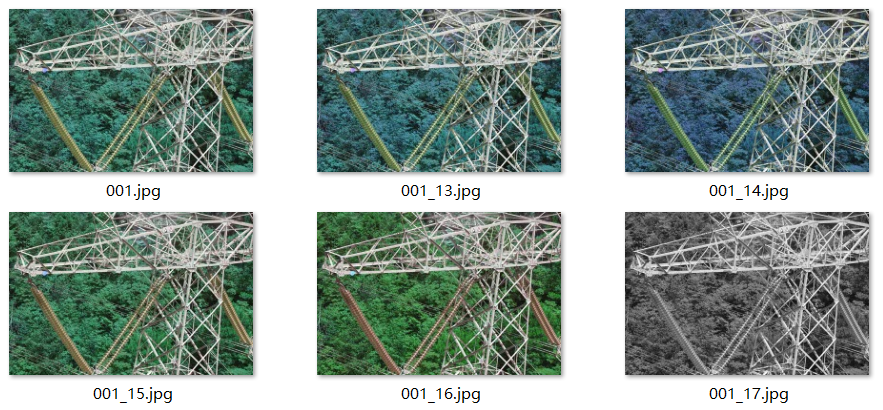


图3 色彩变换得到的图像

经过上述增强后，加上原图，一共得到了19倍的数据，即760张高分辨率图像。由于PSPNet的输入大小为713\*713，所以我们继续对增强后的图片切块，使得每块图像都是713\*713的大小。如果某个子块到图片边界不够713个像素，则补0填充。我们一共得到了4万多个子块图像及其掩模，作为PSPNet的输入进行训练。图4是一张高分辨率图片切块后得到的一些子块图像。具体Python代码的实现可见附录二。



图4 一张高分辨率图片切块后得到的一些子块图像

当然，我们不能把全部样本都用于训练，为了评估训练得到的神经网络的性能，我们把原来的40张高分辨率图像划分训练集和测试集。并且只对训练集进行增强，对增强后的训练集和原始的测试集进行同样的切块，并保存到不同的目录。数据预处理流程如图5所示。

原图

验证集

划分样本

测试集

训练集本

原图

掩模图

图片增强

分块

分块

训练

测试

训练的模型

图5 图像数据预处理流程图

**3 绝缘子语义分割**

利用数据预处理得到的子块图像及其掩模，我们使用金字塔场景解析网络PSPNet进行语义分割。因为FCN不能有效的处理场景之间的关系和全局信息。Zhao等在CVPR2017提出了能够获取全局场景的深度网络PSPNet，能够融合合适的全局特征，将局部和全局信息融合到一起，在多个数据集上表现优异。其网络结构图和具体参数如图6和图7所示。

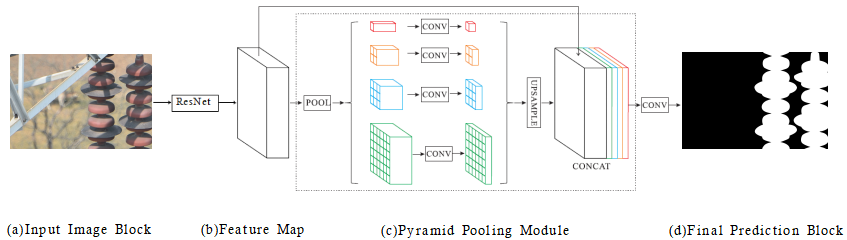


图6 PSPNet的网络结构

|  |  |  |  |
| --- | --- | --- | --- |
| **Layer (type)** | **Output Shape** | **param** | **Connected to** |
| input\_2 (InputLayer) | (None, 713, 713, 3) | 0 |  |
|  | **ResNet51** |  |  |
| activation\_110 (Activation) | (None, 60, 90, 2048) | 0 | add\_32[0][0] |
| average\_pooling2d\_8 (AveragePoo | (None, 6, 9, 2048) | 0 | activation\_110[0][0] |
| average\_pooling2d\_7 (AveragePoo | (None, 3, 4, 2048) | 0 | activation\_110[0][0] |
| average\_pooling2d\_6 (AveragePoo | (None, 2, 3, 2048) | 0 | activation\_110[0][0] |
| average\_pooling2d\_5 (AveragePoo | (None, 1, 1, 2048) | 0 | activation\_110[0][0] |
| PSP\_conv5\_3\_pool6\_conv (Conv2D) | (None, 6, 9, 512) | 1048576 | average\_pooling2d\_8[0][0] |
| PSP\_conv5\_3\_pool3\_conv (Conv2D) | (None, 3, 4, 512) | 1048576 | average\_pooling2d\_7[0][0] |
| PSP\_conv5\_3\_pool2\_conv (Conv2D) | (None, 2, 3, 512) | 1048576 | average\_pooling2d\_6[0][0] |
| PSP\_conv5\_3\_pool1\_conv (Conv2D) | (None, 1, 1, 512) | 1048576 | average\_pooling2d\_5[0][0] |
| PSP\_conv5\_3\_pool6\_conv\_bn (Batc | (None, 6, 9, 512) | 2048 | PSP\_conv5\_3\_pool6\_conv[0][0] |
| PSP\_conv5\_3\_pool3\_conv\_bn (Batc | (None, 3, 4, 512) | 2048 | PSP\_conv5\_3\_pool3\_conv[0][0] |
| PSP\_conv5\_3\_pool2\_conv\_bn (Batc | (None, 2, 3, 512) | 2048 | PSP\_conv5\_3\_pool2\_conv[0][0] |
| PSP\_conv5\_3\_pool1\_conv\_bn (Batc | (None, 1, 1, 512) | 2048 | PSP\_conv5\_3\_pool1\_conv[0][0] |
| activation\_114 (Activation) | (None, 6, 9, 512) | 0 | PSP\_conv5\_3\_pool6\_conv\_bn[0][0] |
| activation\_113 (Activation) | (None, 3, 4, 512) | 0 | PSP\_conv5\_3\_pool3\_conv\_bn[0][0] |
| activation\_112 (Activation) | (None, 2, 3, 512) | 0 | PSP\_conv5\_3\_pool2\_conv\_bn[0][0] |
| activation\_111 (Activation) | (None, 1, 1, 512) | 0 | PSP\_conv5\_3\_pool1\_conv\_bn[0][0] |
| interp\_9 (Interp) | (None, 60, 90, 512) | 0 | activation\_114[0][0] |
| interp\_8 (Interp) | (None, 60, 90, 512) | 0 | activation\_113[0][0] |
| interp\_7 (Interp) | (None, 60, 90, 512) | 0 | activation\_112[0][0] |
| interp\_6 (Interp) | (None, 60, 90, 512) | 0 | activation\_111[0][0] |
| concatenate\_2 (Concatenate) | (None, 60, 90, 4096) | 0 | activation\_110[0][0] |
|  |  |  | interp\_9[0][0] |
|  |  |  | interp\_8[0][0] |
|  |  |  | interp\_7[0][0] |
|  |  |  | interp\_6[0][0] |
| conv5\_4 (Conv2D) | (None, 60, 90, 512) | 18874368 | concatenate\_2[0][0] |
| conv5\_4\_bn (BatchNormalization) | (None, 60, 90, 512) | 2048 | conv5\_4[0][0] |
| activation\_115 (Activation) | (None, 60, 90, 512) | 0 | conv5\_4\_bn[0][0] |
| dropout\_2 (Dropout) | (None, 60, 90, 512) | 0 | activation\_115[0][0] |
| conv6 (Conv2D) | (None, 60, 90, 2) | 1026 | dropout\_2[0][0] |
| interp\_10 (Interp) | (None, 473, 713, 2) | 0 | conv6[0][0] |
| reshape\_2 (Reshape) | (None, 337249, 2) | 0 | interp\_10[0][0] |
| activation\_116 (Activation) | (None, 337249, 2) | 0 | reshape\_2[0][0] |

图7 PSPNet网络的具体参数

对于本题的绝缘子语义分割，因为绝缘子的区域面积相对整幅图像比较小，而且只有一个类别，PSPNet原始的基于交叉熵的损失函数不能很好地提取绝缘子区域，即为了取得包括背景在内的识别率，可能会把一部分绝缘子区域误认为是背景。为了处理这个问题，我们**提出来了一种新损失函数，对绝缘子区域的损失进行了加权，能更完整地提取绝缘子区域。**

原始的PSPNet使用的是普通的交叉熵函数：

其中是像素点的真实类别标签的热编码向量，是样本的网络预测标签。由于绝缘子的区域面积相对整幅图像比较小，即像素点绝大部分都是背景，所以不能很好地描述绝缘子分割这种很不均衡的两类问题。为了处理这个问题，我们提出了一种新的损失函数：

其中是指绝缘子区域的坐标集合。也就是说对绝缘子区域的识别精度进行了加权，当时，将更加关注绝缘子区域。

为了实现这个目的，我们对PSPNet的源代码进行了修改，例如：

* 在model/model\_utils.py文件中，我们把model=Model(img\_input,o)改成了具有双输出的结构 model = Model(img\_input, [o,o])。
* 在 train.py中，我们把:

model.compile(loss=' categorical\_crossentropy')

改为：

model.compile(loss=['categorical\_crossentropy',masked\_categorical\_crossentropy]

其中masked\_categorical\_crossentropy是自定义函数：

**def** masked\_categorical\_crossentropy**(**gt**,** pr**):**

**from** keras**.**losses **import** categorical\_crossentropy

mask **=** 1 **-** gt**[:,** **:,** 0**]**

**return** categorical\_crossentropy**(**gt**,** pr**)** **\*** mask

即对绝缘子区域进行了加权。

网络模型训练过程中的具体参数设置如表1所示。

表1 模型训练主要参数表

|  |  |
| --- | --- |
| 参数 | 参数值 |
| 输入图片尺寸 | 713×713 |
| 类别数（n\_classes） | 2 |
| 最大迭代次数（Epochs） | 300 |
| Batch size | 2 |
| 初始学习率 | 0.01 |
| 学习率衰减策略 | ‘step’ |

表2是PSPNet训练过程的输出。具体Python代码的实现可见附录三。其中，训练300代，每代128个batch，loss是指训练样本的总体交叉熵损失,insulator\_loss是指绝缘子区域的损失，acc是指识别精度，val为前缀的是指验证样本的损失和识别精度。我们可以看到，当迭代到300代时，对训练样本的总体交叉熵损失loss为0.0174，而绝缘子区域的损失insulator\_loss仅仅为0.0028，识别精度acc为0.9952,说明了模型对训练样本拟合得非常好，特别是在绝缘子区域。而对验证样本，识别精度也达到了0.9824。

表2 PSPNet的训练过程

|  |
| --- |
| **Epoch 1/300**  128/128 [==============================] - 123s 964ms/step - loss: 0.8084 - insulator\_loss: 0.1367 - acc: 0.8174 - val\_loss: 1.4985 - val\_insulator\_loss: 0.1555 - val\_acc\_1: 0.6588  Epoch 2/300  128/128 [==============================] - 106s 826ms/step - loss: 0.4684 - insulator\_loss: 0.0865 - acc: 0.8719 - val\_loss: 2.8876 - val\_insulator\_loss: 0.0441 - val\_acc: 0.2823  **Epoch 3/300**  128/128 [==============================] - 107s 835ms/step - loss: 0.3836 - insulator\_loss: 0.0668 - acc: 0.8919 - val\_loss: 1.5488 - val\_insulator\_loss: 0.2329 - val\_acc: 0.7275  **Epoch 4/300**  128/128 [==============================] - 107s 835ms/step - loss: 0.3987 -insulator\_loss: 0.0725 - acc: 0.8971 - val\_loss: 1.5211 - val\_insulator\_loss: 0.1323 - val\_acc: 0.5882  **Epoch 5/300**  128/128 [==============================] - 107s 839ms/step - loss: 0.3355 - softmax\_loss: 0.0594 - acc: 0.9144 - val\_loss: 2.2807 - val\_softmax\_loss: 0.0695 - val\_acc: 0.4635  **...**  **Epoch 100/200**  128/128 [==============================] - 107s 836ms/step - loss: 0.0609 - insulator\_loss: 0.0095 - acc: 0.9828 - val\_loss: 0.7614 - val\_insulator\_loss: 0.1994 - val\_acc: 0.9624  **...**  **Epoch 200/300**  128/128 [==============================] - 107s 835ms/step - loss: 0.0437 - insulator\_loss: 0.0064 - acc: 0.9869 - val\_loss: 0.3245 - val\_insulator\_loss: 0.1051 - val\_acc: 0.9766  **...**  **Epoch 300/300**  128/128 [==============================] - 107s 836ms/step - loss: 0.0174 - insulator\_loss: 0.0028 - acc: 0.9952 - val\_loss: 0.1486 - val\_insulator\_loss: 0.0274 - val\_acc: 0.9824 |

根据上表2的数据统计，我们绘制出PSPNet网络的训练过程参数的变化情况，如下图8所示。

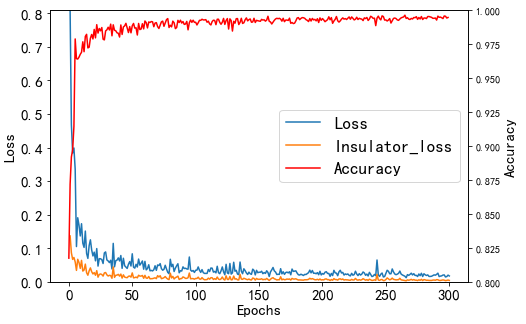


图8 PSPNet网络的训练过程参数的变化情况

图8是PSPNet的训练过程中参数的变化情况。我们可以看到，损失函数loss随着训练迭代次数的增加，损失慢慢变小；准确率softmax\_accuracy随着训练迭代次数的增加，准确慢慢接近1。因此可以说明该方法的效果好。

图9是对测试样本的预测效果图。具体Python代码的实现可见附录四。我们可以看到，我们的模型的预测效果非常好，特别是对绝缘子区域。



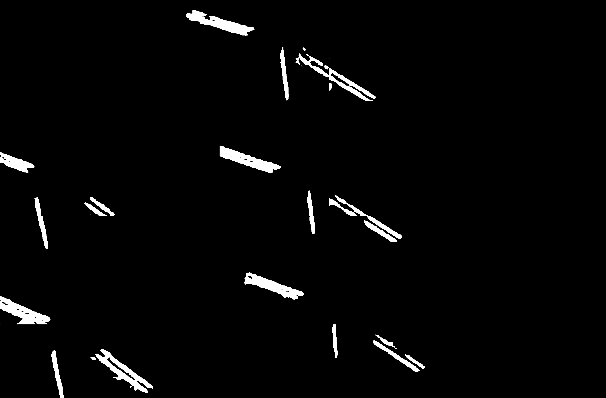
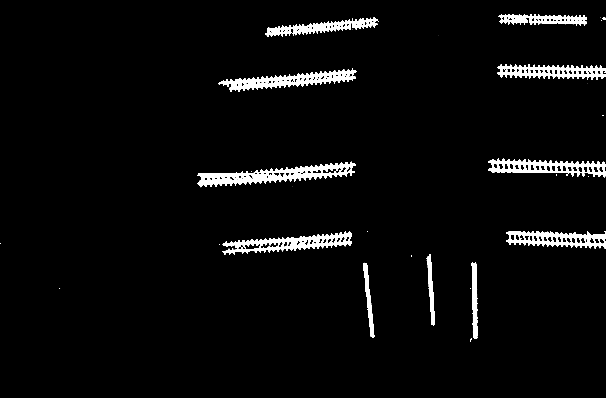


图9 测试样本（上）及其网络模型的预测掩模图（下）

为了进一步得到更好地预测效果，我们将预测除的分割图像进行消除噪音。具体python代码可见附录十。消除噪音后的分割图像如下图10所示。

|  |  |
| --- | --- |
| 019 | 019 |
| 018 | 018 |

图10 原预测掩模图（左）以及过滤噪音后的掩模图（右）

为了进一步说明基于PSPNet方法的绝缘子串珠分割的有效性，我们将采用 Dice[[1]](#footnote-0) 系数进行评价，即评价指标：



本文使用基于原图的标准掩模图像与其相应预测结果掩模图像的两个样本，计算两个样本的相似度，代码可见附录十，其Dice系数的平均值为**0.8474**。因此我们可以看出本文的绝缘子图像分割算法的效果非常好。

**4 绝缘子区域规范化**

提取了绝缘子所在区域的二值掩模之后，我们找出了每个连通区域，并**提出了一种基于主成分的旋转和裁剪算法，**把每串绝缘子单独切分成一个独立的图像，去掉一些噪音，并且旋转到水平方向，规范化为相同的尺寸。Python代码的实现可见附录五。具体步骤如下：

**（1）寻找每一个连通区域。**我们可以直接利用python中的sklearn中的模块获得每个连通区域，图11是找到的连通区域。每个连通区域用不同的颜色进行了划分。

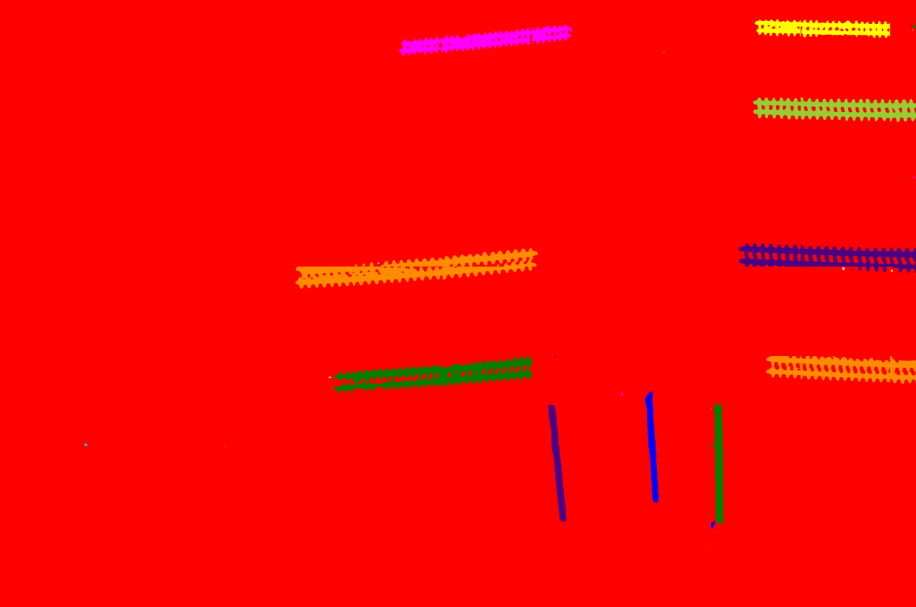


图11 用种子填充算法找到的连通区域

**（2）使用主成分分析进行旋转。**对每一个连通区域，可能包含1个或2个并排的绝缘子串，我们找到其坐标集合，然后计算其协方差矩阵的特征向量，即为该坐标集合的主要方向和次要方向。坐标集合的协方差矩阵为：

其中是均值点的坐标。我们求的两个二维的特征向量和，满足：

令是最大特征根对应的特征向量，则就是坐标集合的主要分布方向。坐标点的主成分分析可用图12解释。设绿色的点表示坐标集合，则就是图中的PCA1的方向，就是PCA2的方向。

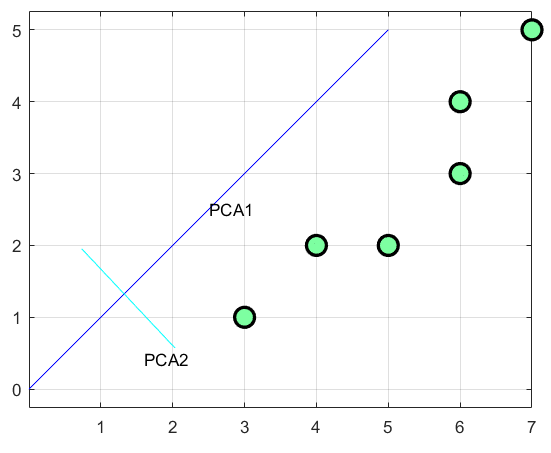


图12 坐标点的主成分分析示意图

**（3）规范化绝缘子图像。**接下来我们把坐标集合中的点在两个特征向量和做投影变换，即把坐标轴旋转到PCA1和PCA2组成的直角坐标系。在新坐标系下，我们可以求得PCA1上的坐标之差的最大值作为长度，PCA2上的坐标之差的最大值作为高度。因为绝缘子串是长条的，**如果，我们认为是噪音，则忽略该连通区域**。

另外，因为绝缘子串通常是两条并排的，它们的掩模很容易形成一个连通区域如图13，此时由于中间有空隙，所以**我们通过直方图来判断是否是两个并排的绝缘子构成的连通区域**。我们计算PCA2上的投影的点的直方图，如图14，我们发现中心有一个明显的波谷。所以，我们可以根据此判断是否是两个并排的绝缘子构成的连通区域。如果是的话，则上下平均切分成两块。

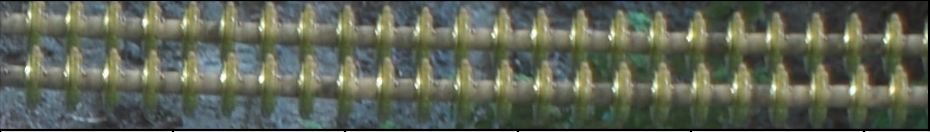


图13 两串并排的绝缘子形成一个连通区域

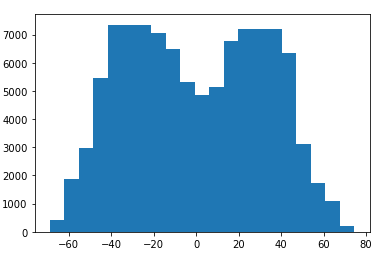


图14 并排两个绝缘子的坐标在第2个主成分上的投影点的直方图

最后，为了便于输入到YOLO模型进行自爆点定位，我们把裁剪出来的绝缘子的高度规范化为128个像素，长度按比例伸缩。然后把把长规范化2048个像素，如果不够2048，则两边补0。如图15是一些绝缘子切割并规范化后得到的图像。





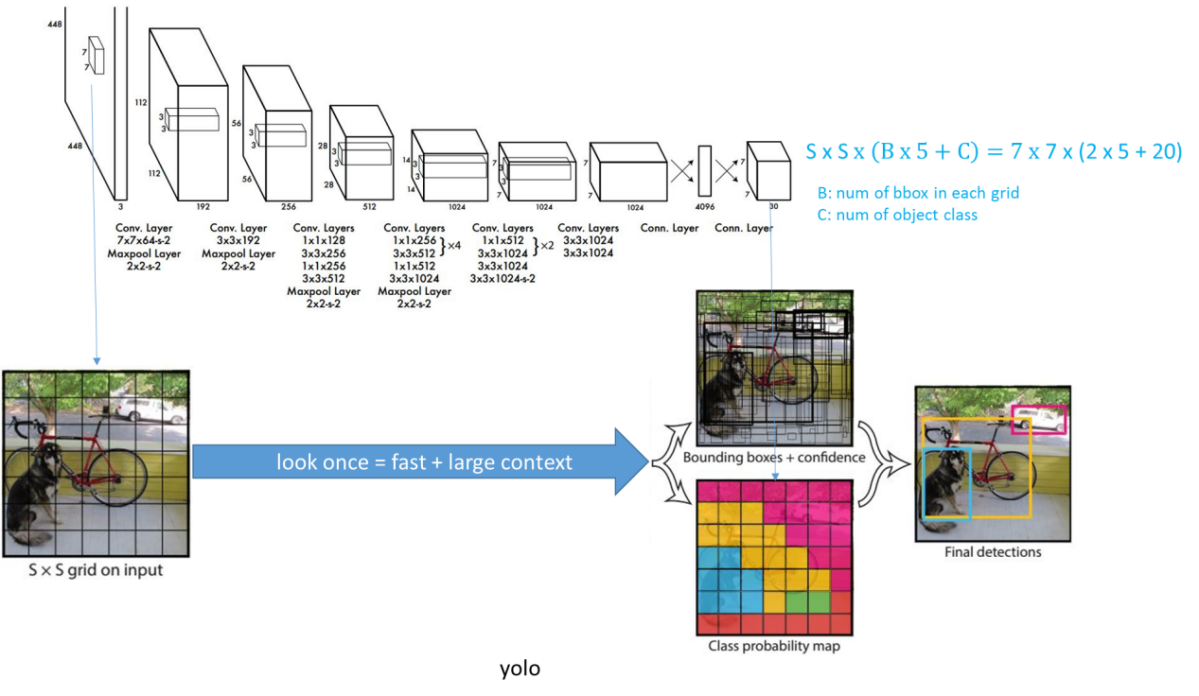
图15一些绝缘子规范化后的图像

**5 自爆点定位**

**（1）YOLO模型。**我们采用的是YOLO V3进行自爆点定位。R-CNN需要数千个单个图像,局限性比较大。而YOLO可以通过单个网络评估来进行预测，这使其变得非常快，比R-CNN快1000倍以上，甚至比[Fast R-CNN快](https://github.com/rbgirshick/fast-rcnn)100倍。这是我们选择YOLO的缘由。

对于传统的滑动窗块的目标检测算法，YOLO算法很好的解决了无法确定被检测目标大小规模而无法确定滑动的步长这一缺点。YOLO将原始图片切割成不重合的小方块，然后通过卷积产生规定大小的特征图，而特征图中的每个元素便是原始图片的小方格，每个元素可以预测那些中心点在该小方格内的目标。

我们把每串绝缘子的规范化图像及其自爆点区域作为YOLO网络模型的输入进行训练，由于图像样本非常单一，**我们利用K-means算法那设计了并指定了几种锚框**，网络模型就能够以非常高的精度定位自爆点的位置。YOLO网络结构图以及以及其卷积结构与残差组成图如图16所示。



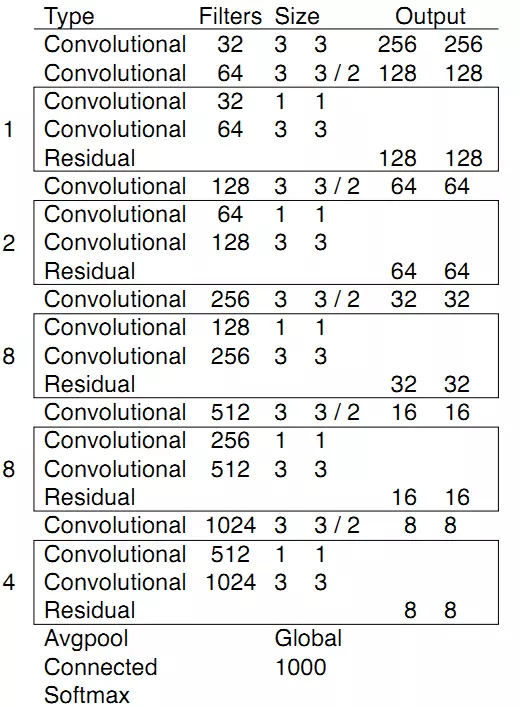


图16 YOLO网络结构图（上）以及其卷积结构与残差组成图（下）

**（2）训练数据生成。**我们为了方便YOLO 对每串绝缘子的规范化图像的数据获取，我们在规范化每张图片的同时获取到每张规范化的图片路径和其对应坐标，并存放到一个train.txt的文本文件中。train.txt的部分数据实例如表13所示。train.txt坐标处为 xmin,ymin,xmax,ymax,id,坐标没有则不用填。对于anchor box，我们**设计了锚框,**YOLO只会对锚框（anchor）中宽高的数值进行搜索。所以锚框的数据由多组高和宽为组成，这里高和宽是我们利用sklearn中的K-means算法对train.txt中数据进行了聚类操作，一共聚成了9类，并获取其聚类中心所得。其训练所用的绝缘子规范化图像及其自爆点坐标的部分数据及anchor数据如下表3所示。

表3 训练所用的绝缘子规范化图像及其自爆点坐标的部分数据及anchor数据

|  |  |
| --- | --- |
| 图片所在路径 | 原始/增强 图片的自爆绝缘子的坐标和id |
| dataset/val\_predict\_pca/037\_4\_0.jpg | 1767,0,1931,128,0 |
| dataset/val\_predict\_pca/038\_1\_0.jpg | 1160,0,1318,128,0 |
| dataset/val\_predict\_pca/039\_3\_0.jpg | 433,0,560,128,0 1004,0,1139,128,0 |
| dataset/val\_predict\_pca/040\_1\_0.jpg | 1074,0,1176,128,0 |
| dataset/val\_predict\_pca/001\_2\_0\_0.jpg | 1470,0,1597,128,0 |
| dataset/val\_predict\_pca/001\_2\_0\_1.jpg | 1470,0,1597,128,0 |
| dataset/val\_predict\_pca/001\_2\_0\_2.jpg | 1470,0,1597,128,0 |
| dataset/val\_predict\_pca/001\_2\_0\_3.jpg | 1470,0,1597,128,0 |
| dataset/val\_predict\_pca/001\_2\_0\_4.jpg | 1470,0,1597,128,0 |
| dataset/val\_predict\_pca/002\_1\_0.jpg |  |
| dataset/val\_predict\_pca/003\_1\_0.jpg |  |
| dataset/val\_predict\_pca/003\_2\_0.jpg |  |
| dataset/val\_predict\_pca/006\_2\_0.jpg | 1115,0,1294,128,0 |
| dataset/val\_predict\_pca/007\_1\_0.jpg | 339,0,529,128,0 |
| dataset/val\_predict\_pca/007\_1\_1.jpg |  |
| dataset/val\_predict\_pca/007\_2\_0.jpg |  |
| dataset/val\_predict\_pca/007\_2\_1.jpg |  |
| dataset/val\_predict\_pca/008\_1\_0.jpg | 649,0,836,128,0 |
| dataset/val\_predict\_pca/009\_5\_0.jpg | 544,0,707,128,0 |
| dataset/val\_predict\_pca/010\_4\_0.jpg | 544,0,706,128,0 |
| dataset/val\_predict\_pca/011\_20\_0.jpg | 649,0,848,128,0 |
| dataset/val\_predict\_pca/011\_27\_0.jpg | 547,0,711,128,0 |
| dataset/val\_predict\_pca/012\_4\_0.jpg | 544,0,703,128,0 |
| dataset/val\_predict\_pca/013\_1\_0.jpg | 669,0,846,128,0 |
| dataset/val\_predict\_pca/014\_4\_0.jpg | 593,0,790,128,0 |
| … | … |
| 锚框数据（分别是模型搜索时需要搜索的：高,宽） | 128,128, 128,96, 128,160, 128,108, 108,108, 108,142, 128,172, 128,142, 128,192 |

**（3）YOLO的训练过程。**我们迭代了80次，每一次的batch为50。具体Python代码的实现可见附录六。其训练过程如下表4所示。

表4 YOLO训练过程

|  |
| --- |
| **Epoch 1/80**  50/50 [==============================] - 55s 1s/step - loss: 8961.2114 - val\_loss: nan  **Epoch 2/80**  50/50 [==============================] - 44s 880ms/step - loss: 3458.1783 - val\_loss: nan  **Epoch 3/80**  50/50 [==============================] - 45s 902ms/step - loss: 805.0133 - val\_loss: nan  **Epoch 4/80**  50/50 [==============================] - 45s 902ms/step - loss: 309.4423 - val\_loss: nan  **Epoch 5/80**  50/50 [==============================] - 46s 917ms/step - loss: 173.7631 - val\_loss: nan  ……  **Epoch 75/80**  50/50 [==============================] - 46s 919ms/step - loss: 5.9625 - val\_loss: 7.0242  **Epoch 76/80**  50/50 [==============================] - 46s 915ms/step - loss: 5.8536 - val\_loss: 5.1144  **Epoch 77/80**  50/50 [==============================] - 46s 919ms/step - loss: 5.7419 - val\_loss: 5.7852  **Epoch 78/80**  50/50 [==============================] - 46s 920ms/step - loss: 6.2527 - val\_loss: 4.6360  **Epoch 79/80**  50/50 [==============================] - 46s 922ms/step - loss: 5.5180 - val\_loss: 4.9174  **Epoch 80/80**  50/50 [==============================] - 46s 924ms/step - loss: 5.4152 - val\_loss: 5.2333 |

由表4可知，一开始val\_loss出现了nan是因为有些绝缘子没有自爆的坐标，后面val\_loss逐渐出现且收敛，并且loss一直变小，最终loss达到5.4152， val\_loss达到5.2333，已经达到足够小的交叉熵损失，由这可以知道最终得到效果非常好的模型。

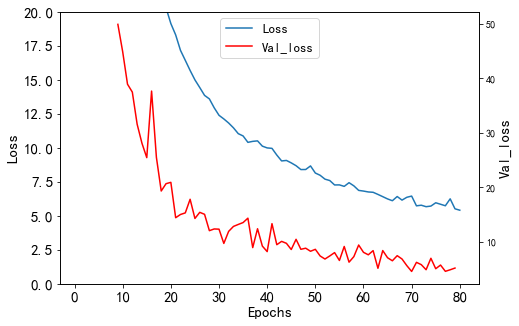


图17 YOLO模型的训练过程

由上表数据可得训练误差变化图如图17，由图可以看出该模型对数据处理收敛的非常快，最终loss与val\_loss达到了比较小的值，效果很好。

**（4）通过训练好的模型预测绝缘子的坐标。**将同样规范化的绝缘子图片通过已经训练好的模型进行预测，具体Python代码的实现可见附录七，得到很好的预测效果（蓝色框为示例绝缘子自爆点位置框，红色框为自爆点预测效果框），如下图18所示。

坐标(1459.2075, 32.697155)(1618.2595,119.41858)



坐标：(993.02313, 1.0119019) (1174.6034, 127.18167)



坐标：(637.47015, 0.92533875) (842.4069, 127.16052)



坐标：(543.4292, 0.7329216) (726.8268, 127.487625)



坐标：(655.2661, 1.2352142) (856.5193, 126.900375)



图18 YOLO网络模型的预测效果图

1. **通过反平移将预测规范化图片的坐标还原至原图坐标。**我们根据对每张图片规范化的过程记录下来，最后再将每张规范化的图片按照规范化的逆过程执行即可还原到原坐标。还原的原坐标的绝缘子自爆点标记图，如下图19所示。

|  |  |
| --- | --- |
| 004 | 002 |
| 016 | 025 |

图19 还原的原坐标的绝缘子自爆点标记图

我们将采用 IOU系数对绝缘子自爆区域的准确性进行评价，IOU定位为：



我们直接采用规范化图片后的标准坐标及预测规范化图片后得到的坐标进行IOU评估，代码可见附录十一，IOU的平均值为**0.8749**。效果非常好。

除了上述方法，我们还尝试过直接将原图放到YOLO中训练，但是过程中出现了不可完成的缺陷，即机器的的显存不足而导致根本性错误，即使我们将batch size降到最低亦是如此。显然YOLO的显存开销成本是非常大的。当然，我们也尝试将图片的分辨率降低，从而降低显存开销成本进行训练，但效果并不容乐观。对此，我们认为使用YOLO无法直接对原图进行训练和预测。也就是说，我们提出的上述方法进行自爆点的定位是可行有效的。

**6 结论**

训练影像预处理

基于PSPNet的训练样本的子块训练

绝缘子模型生成

（验证样本的子块验证）

绝缘子模型训练

无人机影像+训练

语义分割模型

绝缘子分割识别

绝缘子矩形区域提取

绝缘子识别提取

绝缘子连通区域，train.txt文本数据预处理

绝缘子自爆区域定位标记

Yolo v3训练

利用模型得到测试集自爆绝缘子坐标

图20 绝缘子识别及缺陷检测算法的流程图

在PSPNet和YOLO网络模型的基础上，本文设计了一套基于深度学习与主成分分析的绝缘子自爆故障检测的智能算法，其算法流程图如20所示。本文的主要创新性有以下三点：

* 提出来了一种新损失函数对PSPNet进行训练，对绝缘子区域的损失进行了加权，能更完整地提取绝缘子区域。
* 提出了一种基于主成分的旋转和裁剪算法，把每串绝缘子单独切分成一个独立的图像。
* 在YOLO网络的基础上设计并生成了训练数据和锚框数据，能够很好地针对绝缘子自爆点定位这一问题的特点。

实验表明，本文的方法能够在不同光照条件、不同拍摄角度以及复杂背景噪声干扰下的航拍图像中实现绝缘子串的识别与分割，且处理时间短、精度高、鲁棒性强。

**参考文献**

1. 王银立,闫斌.基于视觉的绝缘子“掉串”缺陷的检测与定位[J].计算机工程与设计,2014,35(02):583-587.
2. 陈庆,闫斌,叶润,周小佳.航拍绝缘子卷积神经网络检测及自爆识别研究[J].电子测量与仪器学报,2017,31(06):942-953.
3. 高金峰,吕易航.航拍图像中绝缘子串的识别与分割方法研究[J].郑州大学学报(理学版),2019,51(04):16-22.
4. 陈景文,周鑫,张蓉,张东.基于U-net网络的航拍绝缘子检测[J].陕西科技大学学报,2018,36(04):153-157.
5. 张少平,杨忠,黄宵宁,吴怀群,顾元政.航拍图像中玻璃绝缘子自爆缺陷的检测及定位[J].太赫兹科学与电子信息学报,2013,11(04):609-613.
6. 柳华林,张毅,王海鹏,张立民,李雪腾.一种复杂环境下的胸环靶图分割方法[J/OL].兵器装备工程学报:1-6.
7. S. Ren, K. He, R. Girshick and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, no. 6.
8. OlafRonnerberger,Philipp Ficher,Thomas Brox.U-Net: Convolutional Networks for Biomedical Image Segmentation , in MICCAI 2015 Computer Science ArXivR.
9. E.Shelhamer,J.Long and T.Darrell,Fully Convolutional Networks for Semantic Segmentation,in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol.39.
10. Girshick, "Fast R-CNN," 2015 IEEE International Conference on Computer Vision (ICCV), Santiago, 2015.
11. K. He, G. Gkioxari, P. Dollár and R. Girshick, "Mask R-CNN," 2017 IEEE International Conference on Computer Vision (ICCV), Venice, 2017.
12. K. He, X. Zhang, S. Ren and J. Sun, "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 37, no. 9.
13. X. Zhang, X. Zhou, M. Lin and J. Sun, "ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices," 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, 2018.
14. Zhao H , Shi J , Qi X , et al. Pyramid Scene Parsing Network[C]. IEEE Internetional Conference on Computer Vision and Pattern Recognition (CVPR), 2017.
15. Redmon J , Divvala S , Girshick R , et al. You Only Look Once: Unified, Real-Time Object Detection[C], IEEE Internetional Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

**附 录**

**附录1：图像数据增强main\_autment.py**

'''

对原始的图像数据进行预处理。

对所给图像分别进行放大、缩小、旋转、改变色彩和饱和度等变换，以扩充得到了19倍的图像集。

'''

**import** cv2**,**os

**from** matplotlib **import** pyplot **as** plt

**import** imgaug **as** ia

**from** imgaug **import** augmenters **as** iaa

**import** imageio

**import** numpy **as** np

**import** datetime

# In[data]:

# 原始图像和分割的路径

image\_dir **=** r"dataset\train"

image\_seg\_dir **=** r"dataset\train\_anno"

# 增强后保存图像和分割的路径

image\_aug\_dir **=** r"dataset\aug\_train1"

image\_seg\_aug\_dir **=** r"dataset\aug\_train\_anno1"

**if** **not** os**.**path**.**exists**(**image\_aug\_dir**):**

os**.**makedirs**(**image\_aug\_dir**)**

**if** **not** os**.**path**.**exists**(**image\_seg\_aug\_dir**):**

os**.**makedirs**(**image\_seg\_aug\_dir**)**

# In[set]:

# 变换的集合，每种变换生成一个新的图像样本及其segment

transform\_seqs1 **=** **[**

iaa**.**Affine**(**scale**=**0.6**),**

iaa**.**Affine**(**scale**=**0.8**),**

iaa**.**Affine**(**scale**=**1.2**),**

iaa**.**Affine**(**scale**=**1.4**),**

iaa**.**Fliplr**(),** #0.镜像翻转

iaa**.**Flipud**(),** #1.左右翻转

iaa**.**Affine**(**rotate**=**20**),** #旋转

iaa**.**Affine**(**rotate**=**40**),**

iaa**.**Affine**(**rotate**=**60**),**

iaa**.**Affine**(**rotate**=**90**),**

iaa**.**Affine**(**rotate**=-**20**),**

iaa**.**Affine**(**rotate**=-**40**),**

iaa**.**Affine**(**rotate**=-**60**),**

# 先将图片从RGB变换到HSV,然后将H值增加10,然后再变换回RGB。

iaa**.**WithColorspace**(**to\_colorspace**=**"HSV"**,** from\_colorspace**=**"RGB"**,**

children**=**iaa**.**WithChannels**(**0**,** iaa**.**Add**(**10**))),**

iaa**.**WithColorspace**(**to\_colorspace**=**"HSV"**,** from\_colorspace**=**"RGB"**,**

children**=**iaa**.**WithChannels**(**0**,** iaa**.**Add**(**20**))),**

iaa**.**WithColorspace**(**to\_colorspace**=**"HSV"**,** from\_colorspace**=**"RGB"**,**

children**=**iaa**.**WithChannels**(**0**,** iaa**.**Add**(-**10**))),**

iaa**.**WithColorspace**(**to\_colorspace**=**"HSV"**,** from\_colorspace**=**"RGB"**,**

children**=**iaa**.**WithChannels**(**0**,** iaa**.**Add**(-**20**))),**

iaa**.**Grayscale**()** # 转换成灰度图

**]**

# In[proc]:

start\_time**=**datetime**.**datetime**.**now**()**

image\_set **=** os**.**listdir**(**image\_dir**)**

**for** image\_name **in** image\_set**:**

image **=** cv2**.**imread**(**os**.**path**.**join**(**image\_dir**,**image\_name**))**

image\_name **=** image\_name**[:-**4**]** **+** '.jpg' # 改后缀名

image\_seg\_name **=** image\_name**[:-**4**]** **+** '.png' # 改后缀名

**if** **not** os**.**path**.**exists**(**os**.**path**.**join**(**image\_seg\_dir**,**image\_seg\_name**)):**

**print(**"不存在分割文件："**+**image\_seg\_name**)**

**continue**

image\_seg **=** cv2**.**imread**(**os**.**path**.**join**(**image\_seg\_dir**,**image\_seg\_name**))**

# 把掩模的数据变为只有0和1两类

image\_seg**[**np**.**where**(**image\_seg**>**0**)]=**1

# # 图片太大，先缩小到原来的1/4进行测试

# height,width = int(image.shape[0]/4),int(image.shape[1]/4)

# image = cv2.resize(image, (width, height))

# image\_seg = cv2.resize(image\_seg, (width, height),interpolation=cv2.INTER\_NEAREST)

# 把原始图像也复制进去

imageio**.**imwrite**(**os**.**path**.**join**(**image\_aug\_dir**,**image\_name**),** image**)**

imageio**.**imwrite**(**os**.**path**.**join**(**image\_seg\_aug\_dir**,**image\_seg\_name**),** image\_seg**)**

#将标签转换为SegmentationMapOnImage类型，施加相同变换后能得到正确的标签（不会线性插值）

image\_seg **=** ia**.**SegmentationMapsOnImage**(**image\_seg**,** shape**=**image**.**shape**)**

ind**=**0

**for** ind1**,**tr1 **in** enumerate**(**transform\_seqs1**):**

# if ind1<13:

# continue

image\_aug1 **=** tr1**.**augment\_image**(**image**)**

image\_aug\_seg1 **=** tr1**.**augment\_segmentation\_maps**(**image\_seg**).**get\_arr**().**astype**(**np**.**uint8**)**

ind**=**ind**+**1

image\_aug\_name **=** image\_name**[:-**4**]+**"\_"**+**str**(**ind1**)+**image\_name**[-**4**:]**

image\_seg\_aug\_name **=** image\_seg\_name**[:-**4**]+**"\_"**+**str**(**ind1**)+**image\_seg\_name**[-**4**:]**

imageio**.**imwrite**(**os**.**path**.**join**(**image\_aug\_dir**,**image\_aug\_name**),** image\_aug1**)**

imageio**.**imwrite**(**os**.**path**.**join**(**image\_seg\_aug\_dir**,**image\_seg\_aug\_name**),** image\_aug\_seg1**)**

duration**=**datetime**.**datetime**.**now**()-**start\_time

**print(**'{} done! ------- Duration (s): {}'**.**format**(**image\_name**,** duration**.**seconds**)** **)**

**break**

**附录2：图像分块main\_block.py**

'''

对main\_imgaug.py增强后的训练样本进一步切分成固定大小的子块。

对增强后的训练样本划分小块，713\*713像素为一块，不足的补0。掩模对应切割。

对验证样本也切块，保存到不同目录。

'''

**import** cv2**,**os

**from** matplotlib **import** pyplot **as** plt

**import** imgaug **as** ia

**from** imgaug **import** augmenters **as** iaa

**import** imageio

**import** numpy **as** np

**import** datetime

# In[train]:

# 原始图像和分割的路径

image\_dir **=** r"dataset\aug\_train"

image\_seg\_dir **=** r"dataset\aug\_train\_anno"

# 切分块后保存图像和分割的路径

image\_block\_dir **=** r"dataset\block\_train"

image\_seg\_block\_dir **=** r"dataset\block\_train\_anno"

# In[val]:

## 原始图像和分割的路径

#image\_dir = r"dataset\val"

#image\_seg\_dir = r"dataset\val\_anno"

#

## 切分块后保存图像和分割的路径

#image\_block\_dir = r"dataset\block\_val"

#image\_seg\_block\_dir = r"dataset\block\_val\_anno"

# In[proc]:

# 指定子块的宽和高

block\_width **=** 713

block\_height **=** 713

**if** **not** os**.**path**.**exists**(**image\_block\_dir**):**

os**.**makedirs**(**image\_block\_dir**)**

**if** **not** os**.**path**.**exists**(**image\_seg\_block\_dir**):**

os**.**makedirs**(**image\_seg\_block\_dir**)**

start\_time**=**datetime**.**datetime**.**now**()**

image\_set **=** os**.**listdir**(**image\_dir**)**

icount**=**len**(**image\_set**)**

**for** ii**,**image\_name **in** enumerate**(**image\_set**):**

image **=** cv2**.**imread**(**os**.**path**.**join**(**image\_dir**,**image\_name**))**

image\_name **=** image\_name**[:-**4**]** **+** '.jpg' # 改后缀名

image\_seg\_name **=** image\_name**[:-**4**]** **+** '.png' # 掩模图像的后缀名，使得不会被压缩

**if** **not** os**.**path**.**exists**(**os**.**path**.**join**(**image\_seg\_dir**,**image\_seg\_name**)):**

**print(**"不存在分割文件："**+**image\_seg\_name**)**

**continue**

image\_seg **=** cv2**.**imread**(**os**.**path**.**join**(**image\_seg\_dir**,**image\_seg\_name**))**

# 扩展图像，使得能够被 block\_width 整除

height\_pad **=** block\_height **-** np**.**mod**(**image**.**shape**[**0**],**block\_height**)**

width\_pad **=** block\_width **-** np**.**mod**(**image**.**shape**[**1**],**block\_width**)**

height\_count **=** int**(**image**.**shape**[**0**]/**block\_width**)** **+** int**(**height\_pad**>**0**)**

width\_count **=** int**(**image**.**shape**[**1**]/**block\_width**)** **+** int**(**width\_pad**>**0**)**

image**=**np**.**pad**(**image**,[(**0**,**height\_pad**),(**0**,**width\_pad**),(**0**,**0**)])**

image\_seg**=**np**.**pad**(**image\_seg**,[(**0**,**height\_pad**),(**0**,**width\_pad**),(**0**,**0**)])**

# 把掩模的数据变为只有0和1两类

image\_seg**[**np**.**where**(**image\_seg**>**0**)]=**1

#block\_height = int(image.shape[0]/height\_count)

#block\_width = int(image.shape[1]/width\_count)

**for** hi **in** range**(**height\_count**):**

hb **=** hi**\***block\_height # 子块开始的高度下标

**if** hi**==**height\_count**-**1**:**

he **=** image**.**shape**[**0**]**

**else:**

he **=** hb**+**block\_height

**for** wi **in** range**(**width\_count**):**

wb **=** wi**\***block\_width # 子块开始的宽度下标

**if** wi**==**width\_count**-**1**:**

we **=** image**.**shape**[**1**]**

**else:**

we **=** wb**+**block\_width

image\_block **=** image**[**hb**:**he**,**wb**:**we**,:]**

image\_block\_seg **=** image\_seg**[**hb**:**he**,**wb**:**we**,:]**

# 如果掩模没有绝缘子，则不需要保存该子块

#（该操作大大减少了子块数量，但数据不充分会使得PSPNet的训练效果变差）

# bg = np.where(image\_block\_seg>0)

# if len(bg[0])==0:

# continue

image\_block\_name **=** image\_name**[:-**4**]+**"\_h"**+**str**(**hi**)+**"\_w"**+**str**(**wi**)+**image\_name**[-**4**:]**

image\_block\_seg\_name **=** image\_seg\_name**[:-**4**]+**"\_h"**+**str**(**hi**)+**"\_w"**+**str**(**wi**)+**image\_seg\_name**[-**4**:]**

imageio**.**imwrite**(**os**.**path**.**join**(**image\_block\_dir**,**image\_block\_name**),** image\_block**)**

imageio**.**imwrite**(**os**.**path**.**join**(**image\_seg\_block\_dir**,**image\_block\_seg\_name**),** image\_block\_seg**)**

#plt.imshow(image\_block)

#plt.imshow(image\_block\_seg)

duration**=**datetime**.**datetime**.**now**()-**start\_time

**print(**'{}/{}, {} done! ------- Duration (s): {}'**.**format**(**ii**,**icount**,**image\_name**,** duration**.**seconds**)** **)**

# break

**附录3：PSPNet的训练main\_segment.py**

"""

基于PSPNet的绝缘子区域进行语义分割方法（先要用main\_augment.py和main\_block.py进行数据增强和分割）

训练步骤：

第1步：划分训练和验证样本，并保存到不同目录。

第2步：对训练样本增强，并保存到不同目录。

第3步：对增强后的训练样本划分小块，713\*713像素为一块，不足的补0。掩模对应切割。对验证样本也切块，保存到不同目录。

第4步：基于PSPNet，用训练样本的子块训练，用验证样本的子块验证。得到训练好的语义分割模型。

测试步骤：

对某个测试样本的原图，切分子块，输入到PSPNet中，对每个子块分别得到掩模，然后组装成一个测试样本完整的掩模。

"""

**from** keras\_segmentation**.**train **import** find\_latest\_checkpoint

**from** keras\_segmentation**.**models**.**model\_utils **import** transfer\_weights

**from** keras\_segmentation**.**pretrained **import** pspnet\_50\_ADE\_20K**,**resnet\_pspnet\_VOC12\_v0\_1

**from** keras\_segmentation**.**models**.**pspnet **import** pspnet\_50 #,resnet50\_pspnet

**from** keras**.**utils **import** plot\_model

**from** keras\_segmentation**.**predict **import** get\_colored\_segmentation\_image

**import** datetime**,**cv2

**from** matplotlib **import** pyplot **as** plt

**import** numpy **as** np

**import** os

# In[train]:

# 只有pspnet才有预训练好的数据，(ADE20k数据库：20K张图像，150类)

# 若没有模型文件，则自动下载（由于下载速度很慢，所以建议先把文件放进相应的目录）

# C:\Users\Administrator\.keras\datasets\pspnet50\_ade20k.h5

pretrained\_model **=** pspnet\_50\_ADE\_20K**()**

# 在pspnet中(keras\_segmentation/models/\_pspnet\_2.py)，

# (input\_height,input\_width)只有(473,473),(713,713)的pooling才有定义

new\_model **=** pspnet\_50**(** n\_classes**=**2**,** input\_height**=**473**,** input\_width**=**473**)**

#new\_model = pspnet\_50( n\_classes=2, input\_height=713, input\_width=713)

#new\_model = pspnet\_50( n\_classes=2, input\_height=473, input\_width=713)

# 对基本Res分类网络的权重固定不训练，训练识别率会下降

#for layer in new\_model.layers:

# layerName=str(layer.name)

# if layerName.startswith("Res\_"):

# layer.trainable=False

#new\_model.summary() # 输出模型的每一层的具体参数（非常多）

#plot\_model(new\_model,to\_file = 'new\_model\_473\_713.png') # 画出模型结构图，并保存成图片

file\_model **=** 'new\_model\_org.h5'

**if** **not** os**.**path**.**exists**(**file\_model**):**

# 模型的系数很多，TitianX都要2分钟。

# 每一层，如果权重的形状相同，则复制（最后一层由于类别不同，则不会复制）

transfer\_weights**(** pretrained\_model **,** new\_model **)** # transfer weights from pre-trained model to your model

new\_model**.**save\_weights**(**file\_model**)**

new\_model**.**load\_weights**(**file\_model**)**

# 根据有标注的训练样本进行训练

# keras\_segmentationd的代码做了修改：

# model/model\_utils.py: model = Model(img\_input, [o,o])

# train.py: model.compile(loss=['categorical\_crossentropy',masked\_categorical\_crossentropy]

start\_time**=**datetime**.**datetime**.**now**()**

new\_model**.**train**(**

train\_images **=** "dataset/block\_train/"**,**

train\_annotations **=** "dataset/block\_train\_anno/"**,**

n\_classes**=**2**,**

verify\_dataset**=False,** # 不用每次都验证图像和掩模的数据是否一致

validate**=True,**

loss\_weight**=**2**,**

val\_images**=**"dataset/block\_val/"**,**

val\_annotations**=**"dataset/block\_val\_anno/"**,**

val\_steps\_per\_epoch**=**32**,**

val\_batch\_size**=**2**,**

checkpoints\_path **=** "new\_model" **,**

load\_weights**=None,** # load\_weights='new\_model.10' 继续训练

batch\_size **=** 2**,** # 默认2。batch\_size太大的话，会导致 GPU 内存不够

steps\_per\_epoch **=** 128**,** # 默认512。每一代用多少个batch, None代表自动分割，即数据集样本数/batch样本数

epochs**=**200**,**

**)**

duration**=**datetime**.**datetime**.**now**()-**start\_time

**print(**'-------- Training time (s): {}'**.**format**(**duration**.**seconds**))**

new\_model\_checkpoint **=** find\_latest\_checkpoint**(**"new\_model"**)**

new\_model**.**load\_weights**(**new\_model\_checkpoint**)**

out **=** new\_model**.**predict\_segmentation**(**

inp**=**"dataset/block\_val/015\_h0\_w5.jpg"**,**

out\_fname**=**"out.png"

**)**

out **=** np**.**array**(**out**).**astype**(**np**.**float32**)** # 转换为浮点型，plt.imshow才认为图像在0到1内，即1是白的。

plt**.**imshow**(**out**)**

**附录4：PSPNet的测试main\_predict.py**

'''

根据main\_segment.py训练好的PSPNet网络模型，对测试样本进行预测其掩模图像。

即对某个测试样本的原图，切分子块，输入到PSPNet中，对每个子块分别得到掩模，然后并组装成一个测试样本完整的掩模。

'''

**import** cv2**,**os

**from** matplotlib **import** pyplot **as** plt

**import** imgaug **as** ia

**from** imgaug **import** augmenters **as** iaa

**import** imageio

**import** numpy **as** np

**import** datetime

**from** keras\_segmentation**.**train **import** find\_latest\_checkpoint

**from** keras\_segmentation**.**models**.**model\_utils **import** transfer\_weights

**from** keras\_segmentation**.**pretrained **import** pspnet\_50\_ADE\_20K

**from** keras\_segmentation**.**models**.**pspnet **import** pspnet\_50 #,resnet50\_pspnet

#from cv2 import imresize

# In[model]:

new\_model **=** pspnet\_50**(** n\_classes**=**2**,** input\_height**=**473**,** input\_width**=**473**)**

file\_model **=** 'new\_model.300'

#file\_model = 'new\_model\_val.50'

new\_model**.**load\_weights**(**file\_model**)**

# In[val 测试集（不用先增强，直接拆分）]:

### 原始图像和分割的路径

image\_dir **=** r"dataset\val"

#

## 切分块后保存图像和分割的路径

image\_block\_dir **=** r"dataset\block\_val1"

image\_predict\_dir **=** r"dataset\val\_predict"

# In[proc]:

# 指定子块的宽和高

block\_width **=** 713

block\_height **=** 713

**if** **not** os**.**path**.**exists**(**image\_block\_dir**):**

os**.**makedirs**(**image\_block\_dir**)**

**if** **not** os**.**path**.**exists**(**image\_predict\_dir**):**

os**.**makedirs**(**image\_predict\_dir**)**

#hw = []

#files\_list = []

start\_time**=**datetime**.**datetime**.**now**()**

image\_set **=** os**.**listdir**(**image\_dir**)**

icount**=**len**(**image\_set**)**

**for** ii**,**image\_name **in** enumerate**(**image\_set**):**

image **=** cv2**.**imread**(**os**.**path**.**join**(**image\_dir**,**image\_name**))**

image\_name **=** image\_name**[:-**4**]** **+** '.png' # 改后缀名

# 扩展图像，使得能够被 block\_width 整除

height\_pad **=** block\_height **-** np**.**mod**(**image**.**shape**[**0**],**block\_height**)**

width\_pad **=** block\_width **-** np**.**mod**(**image**.**shape**[**1**],**block\_width**)**

height\_count **=** int**(**image**.**shape**[**0**]/**block\_width**)** **+** int**(**height\_pad**>**0**)**

width\_count **=** int**(**image**.**shape**[**1**]/**block\_width**)** **+** int**(**width\_pad**>**0**)**

image**=**np**.**pad**(**image**,[(**0**,**height\_pad**),(**0**,**width\_pad**),(**0**,**0**)])**

image\_compose **=** np**.**zeros\_like**(**image**)**

#获取每张图片的行数和列数

# hw.append([height\_count,width\_count])

# files = [] #每张图片的分块集

#block\_height = int(image.shape[0]/height\_count)

#block\_width = int(image.shape[1]/width\_count)

**for** hi **in** range**(**height\_count**):**

hb **=** hi**\***block\_height # 子块开始的高度下标

**if** hi**==**height\_count**-**1**:**

he **=** image**.**shape**[**0**]**

**else:**

he **=** hb**+**block\_height

**for** wi **in** range**(**width\_count**):**

wb **=** wi**\***block\_width # 子块开始的宽度下标

**if** wi**==**width\_count**-**1**:**

we **=** image**.**shape**[**1**]**

**else:**

we **=** wb**+**block\_width

image\_block **=** image**[**hb**:**he**,**wb**:**we**,:]**

image\_block\_name **=** image\_name**[:-**4**]+**"\_h"**+**str**(**hi**)+**"\_w"**+**str**(**wi**)+**image\_name**[-**4**:]**

imageio**.**imwrite**(**os**.**path**.**join**(**image\_block\_dir**,**image\_block\_name**),** image\_block**)**

# 保存切分的子块，形成文件路径，以供PSPNet使用（也可以直接用image\_block变量）

anno\_dir\_path**=**os**.**path**.**join**(**image\_block\_dir**,**image\_block\_name**)**

out\_fname**=None** # 不保存预测的分块，只保存组合的整个seg

#out\_fname=os.path.join(image\_predict\_dir,image\_block\_name)

out **=** new\_model**.**predict\_segmentation**(**inp**=**anno\_dir\_path**,**out\_fname**=**out\_fname**)**

#print(out.shape,image\_compose[hb:he,wb:we,:].shape)

out**=**cv2**.**resize**(**out**,** **(**block\_width**,** block\_height**),**interpolation**=**cv2**.**INTER\_NEAREST**)**

# image\_compose[hb:he,wb:we,0]=out\*255

image\_compose**[**hb**:**he**,**wb**:**we**,**0**]=**out**\***255

image\_compose**[**hb**:**he**,**wb**:**we**,**1**]=**out**\***255

image\_compose**[**hb**:**he**,**wb**:**we**,**2**]=**out**\***255

# image\_compose=image\_compose\*255

# 去掉pad的空白，不然掩模和原图大小不一致，在绝缘子规范化时main\_predict\_pca.py会有问题。

image\_compose1 **=** image\_compose**[:-**height\_pad**,:-**width\_pad**,:]**

imageio**.**imwrite**(**os**.**path**.**join**(**image\_predict\_dir**,**image\_name**),** image\_compose1**)**

duration**=**datetime**.**datetime**.**now**()-**start\_time

**print(**'{}/{}, {} done! ------- Duration (s): {}'**.**format**(**ii**,**icount**,**image\_name**,** duration**.**seconds**)** **)**

# break

**附录5：基于主成分的旋转和裁剪的规范化绝缘子区域算法main\_predict\_pca.py**

'''

基于主成分的旋转和裁剪的规范化绝缘子区域算法。

用PSPNet提取了绝缘子所在区域的二值掩模之后，

找出每个连通区域（1个连通区域可能包含1个或2个并排的绝缘子串）的坐标集合，

求连通区域的坐标的均值点，和坐标点集合的PCA的两个方向，求坐标集合在分别在两个方向上的投影。

得到投影的最小值和最大值的差，即可得到连通区域的长度和宽度（若长度和宽度之比小于3，则认为是噪音，忽略）。

把绝缘子串旋转到水平方向，规范化为相同的尺寸128×2048。

把每串绝缘子单独切分成一个独立的图像。

得到了每串绝缘子规范化的图片之后，还要标记自爆点的坐标（若存在的话）

并形成train.txt文件，以供YOLO模型使用。

'''

**import** cv2**,**os**,**imutils

**from** matplotlib **import** pyplot **as** plt

**import** imgaug **as** ia

**from** imgaug **import** augmenters **as** iaa

**import** imageio

**import** numpy **as** np

**import** datetime

**from** skimage **import** measure**,** color

**from** sklearn**.**decomposition **import** PCA

# In[data]:

### 原始图像的路径

image\_dir **=** r"dataset\val"

#image\_dir = r"dataset\train"

#

## 相应掩模的路径

image\_predict\_dir **=** r"dataset\val\_anno" # 训练时，可以直解使用已知的掩模

#image\_predict\_dir = r"dataset\val\_predict" # 测试时，用模型预测得到的掩模

#image\_predict\_dir = r"dataset\train\_anno\_proc" # 把相交的绝缘子先切开

# 绝缘子区域的长方形子块保存的路径（若没有该目录，则自动创建）

image\_predict\_pca\_dir **=** r"dataset\val\_predict\_pca"

# In[proc]:

**if** **not** os**.**path**.**exists**(**image\_predict\_pca\_dir**):**

os**.**makedirs**(**image\_predict\_pca\_dir**)**

start\_time**=**datetime**.**datetime**.**now**()**

image\_set **=** os**.**listdir**(**image\_dir**)**

icount**=**len**(**image\_set**)**

**for** ii**,**image\_name **in** enumerate**(**image\_set**):**

# image\_name='002.jpg'

image **=** cv2**.**imread**(**os**.**path**.**join**(**image\_dir**,**image\_name**))**

image\_name **=** image\_name**[:-**4**]** **+** '.jpg' # 改后缀名为小写

image\_seg\_name **=** image\_name**[:-**4**]** **+** '.png' # 掩模的后缀名为png，不要用jpg压缩

**if** **not** os**.**path**.**exists**(**os**.**path**.**join**(**image\_predict\_dir**,**image\_seg\_name**)):**

**print(**"不存在分割文件："**+**image\_seg\_name**)**

**continue**

image\_seg **=** cv2**.**imread**(**os**.**path**.**join**(**image\_predict\_dir**,**image\_seg\_name**))**

image\_seg **=** image\_seg**[:,:,**0**]** # 只取一个分量即可

image\_seg**[**image\_seg**>**0**]=**1 # 原始数据的掩模，非0的有很多种数，而模型的输入要求是二值的

# # 测试：原图缩小一半

# height,width = int(image.shape[0]/2), int(image.shape[1]/2)

# image=cv2.resize(image, (width, height))

# image\_seg=cv2.resize(image\_seg, (width, height),interpolation=cv2.INTER\_NEAREST)

# 采用skimage中的measure，寻找每一个连通区域

labeled\_img**,** num **=** measure**.**label**(**image\_seg**,** background**=**0**,** return\_num**=True)**

dst **=** color**.**label2rgb**(**labeled\_img**)**

# plt.figure(dpi=150)

# plt.imshow(dst)

# imageio.imwrite('dst.jpg', dst)

# break

classes **=** np**.**unique**(**labeled\_img**)**

classes **=** classes**[**1**:]** # 不要背景0这一类

**for** c **in** classes**:**

inds **=** np**.**where**(**labeled\_img**==**c**)**

inds **=** np**.**array**(**inds**).**T

# 忽略太少的点组成的连通区域

**if** len**(**inds**[:,**1**])<**100**:**

**continue**

# imagec = np.zeros\_like(image\_seg)

# imagec[inds[:,0],inds[:,1]] = 255;

# plt.figure(dpi=100)

# plt.imshow(imagec)

# 对该连通区域所有点的坐标集合，进行PCA变换

trans\_pca **=** PCA**(**n\_components**=**2**).**fit**(**inds**)**

pcas **=** trans\_pca**.**components\_ # PCA的两个主成分，即为点的坐标集合的主要方向和次要方向

# 最主要的方向，计算方向角度theta，准备旋转theta角变成水平方向

pca1 **=** pcas**[**0**]**

sinp **=** pca1**[**0**]/**np**.**linalg**.**norm**(**pca1**)**

cosp **=** pca1**[**1**]/**np**.**linalg**.**norm**(**pca1**)**

theta **=** abs**(**np**.**arcsin**(**sinp**))** # [0, pi/2]

**if** pca1**[**0**]\***pca1**[**1**]>**0**:**

theta**=-**theta

theta **=** theta**\***180**/**np**.**pi #将弧度制转为角度制

inds\_pca **=** trans\_pca**.**transform**(**inds**)**

# plt.figure()

# plt.scatter(inds\_pca[:,0],inds\_pca[:,1],marker='.')

# pca1上的投影坐标之差的最大值作为长度tl，pca2上的坐标之差的最大值作为高度tw

**[**tl**,**tw**]** **=** np**.**max**(**inds\_pca**,**axis**=**0**)** **-** np**.**min**(**inds\_pca**,**axis**=**0**)**

# lower\_q=np.quantile(inds\_pca[:,1],0.05) # 下分位数，2%的位置，去除噪音

# uper\_q =np.quantile(inds\_pca[:,1],0.95) # 上分位数

# tw=uper\_q-lower\_q

# tw=tw\*1.3

# 连通区域的长度和宽度之比，小于3，则认为是噪音

**if** tl**<**100 **or** tw**==**0 **or** tl**/**tw**<**3**:**

**continue**

# 统计直方图，如果最大bin的点的数量显著大于中间bin的点的数量，则需要分为两串绝缘子

#plt.figure()

counts**,**binc**,**aa**=**plt**.**hist**(**inds\_pca**[:,**1**],**21**)**

#break

**if** np**.**max**(**counts**)** **-** counts**[**10**]** **>** np**.**max**(**counts**)/**20**:**

incount **=** 2

tw **=** tw**/**2

i1 **=** np**.**where**(**inds\_pca**[:,**1**]>**0**)[**0**]** # 第1串绝缘子的点的下标

i2 **=** np**.**where**(**inds\_pca**[:,**1**]<=**0**)[**0**]** # 第2串绝缘子的点的下标

inds1**=**inds**[**i1**,:]**

inds2**=**inds**[**i2**,:]**

**else:**

incount **=** 1

inds1**=**inds

inds2**=**inds

tw **=** int**(**tw**)** # 绝缘子串的宽度，取整

**for** ini **in** range**(**incount**):**

**if** ini**==**0**:** inds**=**inds1

**else:** inds**=**inds2

inds\_mean **=** np**.**mean**(**inds**,**axis**=**0**)** # 质心

maxs **=** np**.**max**(**inds**,**axis**=**0**)**

mins **=** np**.**min**(**inds**,**axis**=**0**)**

# 在原始图像上，裁剪出一个绝缘子所在的矩形区域

image1 **=** image**[**mins**[**0**]:**maxs**[**0**],**mins**[**1**]:**maxs**[**1**],:]**

#plt.figure(dpi=200)

#plt.imshow(image1)

# 把矩形区域进行旋转theta角度，使得绝缘子水平放置

image1\_rotated **=** imutils**.**rotate\_bound**(**image1**,** theta**)**

# plt.figure(dpi=200)

# plt.imshow(image1\_rotated)

# 继续把上下多余的部分裁剪掉

hh**,**ww **=** image1\_rotated**.**shape**[**0**],**image1\_rotated**.**shape**[**1**]**

aa **=** int**((**hh**-**tw**)/**2**)**

**if** aa**>**0**:**

image1\_rotated **=** image1\_rotated**[**aa**:-**aa**,:,:]**

# 高度缩放到128

width **=** int**(**128**\***image1\_rotated**.**shape**[**1**]/**image1\_rotated**.**shape**[**0**])**

image1\_rotated1**=**cv2**.**resize**(**image1\_rotated**,** **(**width**,**128**))**

# 宽度缩放到2048或者两边补0

**if** width **>** 2048**:**

width**=**2048

image\_box**=**cv2**.**resize**(**image1\_rotated**,** **(**2048**,**128**))**

**else:**

pad **=** int**((**2048**-**width**)/**2**)**

image\_box **=** np**.**pad**(**image1\_rotated1**,[(**0**,**0**),(**pad**,**pad**),(**0**,**0**)])**

image\_box\_name **=** image\_name**[:-**4**]+**"\_"**+**str**(**c**)+**'\_'**+**str**(**ini**)+**image\_name**[-**4**:]**

imageio**.**imwrite**(**os**.**path**.**join**(**image\_predict\_pca\_dir**,**image\_box\_name**),** image\_box**)**

# break

# break

duration**=**datetime**.**datetime**.**now**()-**start\_time

**print(**'{}/{}, {} done! ------- Duration (s): {}'**.**format**(**ii**,**icount**,**image\_name**,** duration**.**seconds**)** **)**

**break**

**附录6：YOLO模型的训练main\_yolo\_train.py**

"""

Retrain the YOLO model for your own dataset.

"""

**import** numpy **as** np

**import** keras**.**backend **as** K

**from** keras**.**layers **import** Input**,** Lambda

**from** keras**.**models **import** Model

**from** keras**.**callbacks **import** TensorBoard**,** ModelCheckpoint**,** EarlyStopping

**from** yolo3**.**model **import** preprocess\_true\_boxes**,** yolo\_body**,** tiny\_yolo\_body**,** yolo\_loss

**from** yolo3**.**utils **import** get\_random\_data

**def** \_main**():**

annotation\_path **=** 'train.txt'

log\_dir **=** 'logs/000/'

classes\_path **=** 'model\_data/voc\_classes.txt'

anchors\_path **=** 'model\_data/yolo\_anchors.txt'

class\_names **=** get\_classes**(**classes\_path**)**

anchors **=** get\_anchors**(**anchors\_path**)**

input\_shape **=** **(**128**,**2048**)** # multiple of 32, hw

model **=** create\_model**(**input\_shape**,** anchors**,** **len(**class\_names**)** **)**

train**(**model**,** annotation\_path**,** input\_shape**,** anchors**,** **len(**class\_names**),** log\_dir**=**log\_dir**)**

**def** train**(**model**,** annotation\_path**,** input\_shape**,** anchors**,** num\_classes**,** log\_dir**=**'logs/'**):**

model**.compile(**optimizer**=**'adam'**,** loss**={**

'yolo\_loss'**:** **lambda** y\_true**,** y\_pred**:** y\_pred**})**

logging **=** TensorBoard**(**log\_dir**=**log\_dir**)**

checkpoint **=** ModelCheckpoint**(**log\_dir **+** "ep{epoch:03d}-loss{loss:.3f}-val\_loss{val\_loss:.3f}.h5"**,**

monitor**=**'val\_loss'**,** save\_weights\_only**=True,** save\_best\_only**=True,** period**=**1**)**

batch\_size **=** 8

val\_split **=** 0.1

# print(checkpoint)

**with** **open(**annotation\_path**)** **as** f**:**

lines **=** f**.**readlines**()**

np**.**random**.**seed**(**10101**)**

np**.**random**.**shuffle**(**lines**)**

np**.**random**.**seed**(None)**

num\_val **=** **int(len(**lines**)\***val\_split**)**

num\_train **=** **len(**lines**)** **-** num\_val

**print(**'Train on {} samples, val on {} samples, with batch size {}.'**.format(**num\_train**,** num\_val**,** batch\_size**))**

model**.**fit\_generator**(**data\_generator\_wrapper**(**lines**[:**num\_train**],** batch\_size**,** input\_shape**,** anchors**,** num\_classes**),**

# steps\_per\_epoch=max(1, num\_train//batch\_size),

steps\_per\_epoch**=**50**,**

validation\_data**=**data\_generator\_wrapper**(**lines**[:**num\_train**],** batch\_size**,** input\_shape**,** anchors**,** num\_classes**),**

validation\_steps**=max(**1**,** num\_val**//**batch\_size**),**

epochs**=**120**,**

initial\_epoch**=**0**,**

callbacks**=[**checkpoint**])**

model**.**save\_weights**(**log\_dir **+** 'trained\_weights.h5'**)**

**def** get\_classes**(**classes\_path**):**

**with** **open(**classes\_path**)** **as** f**:**

class\_names **=** f**.**readlines**()**

class\_names **=** **[**c**.**strip**()** **for** c **in** class\_names**]**

**return** class\_names

**def** get\_anchors**(**anchors\_path**):**

**with** **open(**anchors\_path**)** **as** f**:**

anchors **=** f**.**readline**()**

anchors **=** **[float(**x**)** **for** x **in** anchors**.**split**(**','**)]**

**return** np**.**array**(**anchors**).**reshape**(-**1**,** 2**)**

**def** create\_model**(**input\_shape**,** anchors**,** num\_classes**,** load\_pretrained**=True,** freeze\_body**=False,**

weights\_path**=**'model\_data/yolo\_weights.h5'**):**

K**.**clear\_session**()** # get a new session

image\_input **=** Input**(**shape**=(None,** **None,** 3**))**

h**,** w **=** input\_shape

num\_anchors **=** **len(**anchors**)**

y\_true **=** **[**Input**(**shape**=(**h**//{**0**:**32**,** 1**:**16**,** 2**:**8**}[**l**],** w**//{**0**:**32**,** 1**:**16**,** 2**:**8**}[**l**],** \

num\_anchors**//**3**,** num\_classes**+**5**))** **for** l **in** **range(**3**)]**

model\_body **=** yolo\_body**(**image\_input**,** num\_anchors**//**3**,** num\_classes**)**

**print(**'Create YOLOv3 model with {} anchors and {} classes.'**.format(**num\_anchors**,** num\_classes**))**

**if** load\_pretrained**:**

model\_body**.**load\_weights**(**weights\_path**,** by\_name**=True,** skip\_mismatch**=True)**

**print(**'Load weights {}.'**.format(**weights\_path**))**

**if** freeze\_body**:**

# Do not freeze 3 output layers.

num **=** **len(**model\_body**.**layers**)-**7

**for** i **in** **range(**num**):** model\_body**.**layers**[**i**].**trainable **=** **False**

**print(**'Freeze the first {} layers of total {} layers.'**.format(**num**,** **len(**model\_body**.**layers**)))**

model\_loss **=** Lambda**(**yolo\_loss**,** output\_shape**=(**1**,),** name**=**'yolo\_loss'**,**

arguments**={**'anchors'**:** anchors**,** 'num\_classes'**:** num\_classes**,** 'ignore\_thresh'**:** 0.5**})(**

**[\***model\_body**.**output**,** **\***y\_true**])**

model **=** Model**([**model\_body**.**input**,** **\***y\_true**],** model\_loss**)**

**return** model

**def** data\_generator**(**annotation\_lines**,** batch\_size**,** input\_shape**,** anchors**,** num\_classes**):**

n **=** len**(**annotation\_lines**)**

np**.**random**.**shuffle**(**annotation\_lines**)**

i **=** 0

**while** **True:**

image\_data **=** **[]**

box\_data **=** **[]**

**for** b **in** range**(**batch\_size**):**

i **%=** n

image**,** box **=** get\_random\_data**(**annotation\_lines**[**i**],** input\_shape**,** random**=True)**

image\_data**.**append**(**image**)**

box\_data**.**append**(**box**)**

i **+=** 1

image\_data **=** np**.**array**(**image\_data**)**

box\_data **=** np**.**array**(**box\_data**)**

y\_true **=** preprocess\_true\_boxes**(**box\_data**,** input\_shape**,** anchors**,** num\_classes**)**

**yield** **[**image\_data**,** **\***y\_true**],** np**.**zeros**(**batch\_size**)**

**def** data\_generator\_wrapper**(**annotation\_lines**,** batch\_size**,** input\_shape**,** anchors**,** num\_classes**):**

n **=** len**(**annotation\_lines**)**

**if** n**==**0 **or** batch\_size**<=**0**:** **return** **None**

**return** data\_generator**(**annotation\_lines**,** batch\_size**,** input\_shape**,** anchors**,** num\_classes**)**

**if** \_\_name\_\_ **==** '\_\_main\_\_'**:**

\_main**()**

**附录7：YOLO模型的测试main\_yolo\_predict.py**

# In[预测]

#from yolo import YOLO

**from** yolo\_defect **import** YOLO\_Defect

**from** PIL **import** Image

**import** os

#import keras

**import** glob

#import tensorflow as tf

**import** tensorflow**.**compat**.**v1 **as** tf

tf**.**disable\_v2\_behavior**()**

#keras.backend.clear\_session()

tf**.**keras**.**backend**.**clear\_session**()**

FLAGS **=** **{}**

defection **=** YOLO\_Defect**(\*\*(**FLAGS**))**

path **=** "./test\_insulator/\*.JPG"

#outdir = "./result"

valFile **=** **{}**

**for** jpgfile **in** glob**.**glob**(**path**):**

name **=** os**.**path**.**basename**(**jpgfile**)**

img **=** Image**.open(**jpgfile**)**

# img = cv2.imread(jpgfile)

**print(**jpgfile**)**

quexian **=** defection**.**detect\_image**(**img**)**

**print(**quexian**)**

valFile**[**jpgfile**]** **=** quexian

# In[画出图像]

**for** valF **in** valFile**:**

**print(**valF**,**valFile**[**valF**])**

drawbbox**(**valFile**[**valF**],**valF**)**

#drawbbox([[0,1470,128,1597]],"test\_insulator/001\_2\_0\_2.jpg")

# In[画图的函数]

**import** cv2

**import** os

#import matplotlib.pyplot as plt

def drawbbox(points,ImgPath=r"test\_insulator/", savePath=r"dataset/saveVal"):

if not os.path.exists(savePath):

os.makedirs(savePath)

im = cv2.imread(ImgPath)

# xbb = 128 / 100

# ybb = 2048 / 2000

# cv2.polylines(im, [points], True, color=(0, 0, 255),thickness=2)

for point in points:

# if len(points):

print( (point[1],point[0]), (point[3],point[2]) )

cv2.rectangle(im, (point[1],point[0]), (point[3],point[2]), (0, 0, 255),2)

print(os.path.join(savePath, os.path.basename(ImgPath)))

cv2.imwrite(os.path.join(savePath, os.path.basename(ImgPath)), im)

# if imm:

# cv2.imwrite(os.path.join(savePath, os.path.basename(ImgPath)), imm)

**附录8：YOLO模型的测试 (附录7的辅助代码) main\_yolo\_defect.py**

# -\*- coding: utf-8 -\*-

"""

Class definition of YOLO\_v3 style detection model on image and video

"""

**import** colorsys

**import** os

#import keras

**from** timeit **import** default\_timer **as** timer

**import** tensorflow**.**compat**.**v1 **as** tf

#import tensorflow as tfs

**import** numpy **as** np

**from** keras **import** backend **as** K

**from** keras**.**models **import** load\_model

**from** keras**.**layers **import** Input

**from** keras**.**utils **import** multi\_gpu\_model

#from tensorflow.compat.v1.keras import backend as K

#from tensorflow.compat.v1.keras.models import load\_model

#from tensorflow.compat.v1.keras.utils import multi\_gpu\_model

#from tensorflow.compat.v1.keras.layers import Input

**from** PIL **import** Image**,** ImageFont**,** ImageDraw

**from** yolo3**.**model **import** yolo\_eval**,** yolo\_body**,** tiny\_yolo\_body

**from** yolo3**.**utils **import** letterbox\_image

**import** os

**class** **YOLO\_Defect(object):**

\_defaults **=** **{**

#"model\_path": 'model\_data/yolo.h5',

"model\_path"**:** 'model\_data\_defect/ep040-loss10.506-val\_loss8.603.h5'**,**

# "model\_path": 'logs/000/ep040-loss10.506-val\_loss8.603.h5',

"anchors\_path"**:** 'model\_data\_defect/yolo\_anchors.txt'**,**

"classes\_path"**:** 'model\_data\_defect/voc\_classes.txt'**,**

"score" **:** 0.01**,**

"iou" **:** 0.2**,**

"model\_image\_size" **:** **(**128**,** 2048**),**

"gpu\_num" **:** 1**,**

**}**

*@classmethod*

**def** get\_defaults**(**cls**,** n**):**

**if** n **in** cls**.**\_defaults**:**

**return** cls**.**\_defaults**[**n**]**

**else:**

**return** "Unrecognized attribute name '" **+** n **+** "'"

**def** \_\_init\_\_**(**self**,** **\*\***kwargs**):**

self**.**\_\_dict\_\_**.**update**(**self**.**\_defaults**)** # set up default values

self**.**\_\_dict\_\_**.**update**(**kwargs**)** # and update with user overrides

self**.**class\_names **=** self**.**\_get\_class**()**

self**.**anchors **=** self**.**\_get\_anchors**()**

## self.sess = K.get\_session()

self**.**sess **=** tf**.**compat**.**v1**.**keras**.**backend**.**get\_session**()**

# self.sess = tf.compat.v1.Session()

self**.**boxes**,** self**.**scores**,** self**.**classes **=** self**.**generate**()**

**def** \_get\_class**(**self**):**

classes\_path **=** os**.**path**.**expanduser**(**self**.**classes\_path**)**

**with** **open(**classes\_path**)** **as** f**:**

class\_names **=** f**.**readlines**()**

class\_names **=** **[**c**.**strip**()** **for** c **in** class\_names**]**

**return** class\_names

**def** \_get\_anchors**(**self**):**

anchors\_path **=** os**.**path**.**expanduser**(**self**.**anchors\_path**)**

**with** **open(**anchors\_path**)** **as** f**:**

anchors **=** f**.**readline**()**

anchors **=** **[float(**x**)** **for** x **in** anchors**.**split**(**','**)]**

**return** np**.**array**(**anchors**).**reshape**(-**1**,** 2**)**

**def** generate**(**self**):**

model\_path **=** os**.**path**.**expanduser**(**self**.**model\_path**)**

**assert** model\_path**.**endswith**(**'.h5'**),** 'Keras model or weights must be a .h5 file.'

# Load model, or construct model and load weights.

num\_anchors **=** **len(**self**.**anchors**)**

num\_classes **=** **len(**self**.**class\_names**)**

is\_tiny\_version **=** num\_anchors**==**6 # default setting

**try:**

self**.**yolo\_model **=** load\_model**(**model\_path**,** **compile=False)**

**except:**

self**.**yolo\_model **=** tiny\_yolo\_body**(**Input**(**shape**=(None,None,**3**)),** num\_anchors**//**2**,** num\_classes**)** \

**if** is\_tiny\_version **else** yolo\_body**(**Input**(**shape**=(None,None,**3**)),** num\_anchors**//**3**,** num\_classes**)**

self**.**yolo\_model**.**load\_weights**(**self**.**model\_path**)** # make sure model, anchors and classes match

**else:**

**assert** self**.**yolo\_model**.**layers**[-**1**].**output\_shape**[-**1**]** **==** \

num\_anchors**/len(**self**.**yolo\_model**.**output**)** **\*** **(**num\_classes **+** 5**),** \

'Mismatch between model and given anchor and class sizes'

**print(**'{} model, anchors, and classes loaded.'**.format(**model\_path**))**

# Generate colors for drawing bounding boxes.

hsv\_tuples **=** **[(**x **/** **len(**self**.**class\_names**),** 1.**,** 1.**)**

**for** x **in** **range(len(**self**.**class\_names**))]**

self**.**colors **=** **list(map(lambda** x**:** colorsys**.**hsv\_to\_rgb**(\***x**),** hsv\_tuples**))**

self**.**colors **=** **list(**

**map(lambda** x**:** **(int(**x**[**0**]** **\*** 255**),** **int(**x**[**1**]** **\*** 255**),** **int(**x**[**2**]** **\*** 255**)),**

self**.**colors**))**

np**.**random**.**seed**(**10101**)** # Fixed seed for consistent colors across runs.

np**.**random**.**shuffle**(**self**.**colors**)** # Shuffle colors to decorrelate adjacent classes.

np**.**random**.**seed**(None)** # Reset seed to default.

# Generate output tensor targets for filtered bounding boxes.

self**.**input\_image\_shape **=** K**.**placeholder**(**shape**=(**2**,** **))**

**if** self**.**gpu\_num**>=**2**:**

self**.**yolo\_model **=** multi\_gpu\_model**(**self**.**yolo\_model**,** gpus**=**self**.**gpu\_num**)**

boxes**,** scores**,** classes **=** yolo\_eval**(**self**.**yolo\_model**.**output**,** self**.**anchors**,**

**len(**self**.**class\_names**),** self**.**input\_image\_shape**,**

score\_threshold**=**self**.**score**,** iou\_threshold**=**self**.**iou**)**

**return** boxes**,** scores**,** classes

**def** detect\_image**(**self**,** image**):**

start **=** timer**()**

**if** self**.**model\_image\_size **!=** **(None,** **None):**

**assert** self**.**model\_image\_size**[**0**]%**32 **==** 0**,** 'Multiples of 32 required'

**assert** self**.**model\_image\_size**[**1**]%**32 **==** 0**,** 'Multiples of 32 required'

boxed\_image **=** letterbox\_image**(**image**,** **tuple(reversed(**self**.**model\_image\_size**)))**

**else:**

new\_image\_size **=** **(**image**.**width **-** **(**image**.**width **%** 32**),**

image**.**height **-** **(**image**.**height **%** 32**))**

boxed\_image **=** letterbox\_image**(**image**,** new\_image\_size**)**

image\_data **=** np**.**array**(**boxed\_image**,** dtype**=**'float32'**)**

# print(image\_data.shape)

# print('begin detect')

image\_data **/=** 255.

image\_data **=** np**.**expand\_dims**(**image\_data**,** 0**)** # Add batch dimension.

# print('expend success')

out\_boxes**,** out\_scores**,** out\_classes **=** self**.**sess**.**run**(**

**[**self**.**boxes**,** self**.**scores**,** self**.**classes**],**

feed\_dict**={**

self**.**yolo\_model**.input:** image\_data**,**

self**.**input\_image\_shape**:** **[**image**.**size**[**1**],** image**.**size**[**0**]],**

K**.**learning\_phase**():** 0

**})**

**return** out\_boxes

**def** close\_session**(**self**):**

self**.**sess**.**close**()**

**def** detect\_video**(**yolo**,** video\_path**,** output\_path**=**""**):**

**import** cv2

vid **=** cv2**.**VideoCapture**(**video\_path**)**

**if** **not** vid**.**isOpened**():**

**raise** **IOError(**"Couldn't open webcam or video"**)**

video\_FourCC **=** **int(**vid**.**get**(**cv2**.**CAP\_PROP\_FOURCC**))**

video\_fps **=** vid**.**get**(**cv2**.**CAP\_PROP\_FPS**)**

video\_size **=** **(int(**vid**.**get**(**cv2**.**CAP\_PROP\_FRAME\_WIDTH**)),**

**int(**vid**.**get**(**cv2**.**CAP\_PROP\_FRAME\_HEIGHT**)))**

#print(video\_size)

isOutput **=** **True** **if** output\_path **!=** "" **else** **False**

**if** isOutput**:**

**print(**"!!! TYPE:"**,** **type(**output\_path**),** **type(**video\_FourCC**),** **type(**video\_fps**),** **type(**video\_size**))**

out **=** cv2**.**VideoWriter**(**output\_path**,** video\_FourCC**,** video\_fps**,** video\_size**)**

accum\_time **=** 0

curr\_fps **=** 0

fps **=** "FPS: ??"

prev\_time **=** timer**()**

**while** **True:**

return\_value**,** frame **=** vid**.**read**()**

image **=** Image**.**fromarray**(**cv2**.**cvtColor**(**frame**,**cv2**.**COLOR\_BGR2RGB**))**

#print(image.size)

#cv2.namedWindow('a',cv2.WINDOW\_NORMAL)

#cv2.imshow('a',frame)

#print(type(frame))

#image = Image.fromarray(frame)

#frame = cv2.cvtColor(frame,cv2.COLOR\_BGR2RGB)

#cv2.imwrite('aaa/aaa'+'a' + '.jpg', frame)

#image.save('aaa/aaa'+'a' + '.jpg', 'jpeg')

#image = Image.open('aaa/aaa'+'a' + '.jpg')

image **=** yolo**.**detect\_image**(**image**)**

result **=** np**.**asarray**(**image**)**

result **=** cv2**.**cvtColor**(**result**,**cv2**.**COLOR\_RGB2BGR**)**

curr\_time **=** timer**()**

exec\_time **=** curr\_time **-** prev\_time

prev\_time **=** curr\_time

accum\_time **=** accum\_time **+** exec\_time

curr\_fps **=** curr\_fps **+** 1

**if** accum\_time **>** 1**:**

accum\_time **=** accum\_time **-** 1

fps **=** "FPS: " **+** **str(**curr\_fps**)**

curr\_fps **=** 0

cv2**.**putText**(**result**,** text**=**fps**,** org**=(**3**,** 15**),** fontFace**=**cv2**.**FONT\_HERSHEY\_SIMPLEX**,**

fontScale**=**0.50**,** color**=(**255**,** 0**,** 0**),** thickness**=**2**)**

cv2**.**namedWindow**(**"result"**,** cv2**.**WINDOW\_NORMAL**)**

cv2**.**imshow**(**"result"**,** result**)**

**if** isOutput**:**

out**.**write**(**result**)**

**if** cv2**.**waitKey**(**1**)** **&** 0xFF **==** **ord(**'q'**):**

**break**

yolo**.**close\_session**()**

**附录9：消除绝缘子串珠噪音 main\_filter\_noise.py**

**import** cv2**,**os

**from** matplotlib **import** pyplot **as** plt

**import** imgaug **as** ia

**from** imgaug **import** augmenters **as** iaa

**import** imageio

**import** numpy **as** np

**import** datetime

**from** skimage **import** measure**,** color

**from** sklearn**.**decomposition **import** PCA

image\_predict\_dir **=** r"dataset\val\_predict"

image\_predict\_filter\_noise\_dir **=** r"dataset\val\_predict\_filter\_noise"

image\_set**=**os**.**listdir**(**image\_predict\_dir**)**

icount**=len(**image\_set**)**

start\_time**=**datetime**.**datetime**.**now**()**

#开始去掉一些噪音

**for** ii**,**image\_name **in** **enumerate(**image\_set**):**

# image\_name='003.png'

image **=** cv2**.**imread**(**os**.**path**.**join**(**image\_predict\_dir**,**image\_name**))**

image2 **=** image**[:,:,**0**]** # 只取一个分量即可

image2**[**image2**>**0**]=**1 # 原始数据的掩模，非0的有很多种数，而模型的输入要求是二值的

# 采用skimage中的measure，寻找每一个连通区域

labeled\_img**,** num **=** measure**.**label**(**image2**,** background**=**0**,** return\_num**=True)**

dst **=** color**.**label2rgb**(**labeled\_img**)**

classes **=** np**.**unique**(**labeled\_img**)**

classes **=** classes**[**1**:]** # 不要背景0这一类

**for** c **in** classes**:**

inds **=** np**.**where**(**labeled\_img**==**c**)**

inds **=** np**.**array**(**inds**).**T

# 忽略太少的点组成的连通区域

**if** **len(**inds**[:,**1**])<**1500**:**

image2**[**labeled\_img**==**c**]=**0

**else:**

trans\_pca **=** PCA**(**n\_components**=**2**).**fit**(**inds**)**# 对该连通区域所有点的坐标集合，进行PCA变换

pcas **=** trans\_pca**.**components\_ # PCA的两个主成分，即为点的坐标集合的主要方向和次要方向

# 最主要的方向，计算方向角度theta，准备旋转theta角变成水平方向

pca1 **=** pcas**[**0**]**

sinp **=** pca1**[**0**]/**np**.**linalg**.**norm**(**pca1**)**

cosp **=** pca1**[**1**]/**np**.**linalg**.**norm**(**pca1**)**

theta **=** abs**(**np**.**arcsin**(**sinp**))** # [0, pi/2]

**if** pca1**[**0**]\***pca1**[**1**]>**0**:**

theta**=-**theta

theta **=** theta**\***180**/**np**.**pi #将弧度制转为角度制

inds\_pca **=** trans\_pca**.**transform**(**inds**)**

**[**tl**,**tw**]** **=** np**.**max**(**inds\_pca**,**axis**=**0**)** **-** np**.**min**(**inds\_pca**,**axis**=**0**)** # pca1上的投影坐标之差的最大值作为长度tl，pca2上的坐标之差的最大值作为高度tw

# 连通区域的长度和宽度之比，小于3，则认为是噪音

**if** tl**<**500 **or** tw**==**0 **or** tl**/**tw**<**1.5**:**

image2**[**labeled\_img**==**c**]=**0

image**[:,:,**1**]=**image2**\***255

image**[:,:,**2**]=**image2**\***255

image**[:,:,**0**]=**image2**\***255

# plt.imshow(image)

# break

imageio**.**imwrite**(**os**.**path**.**join**(**image\_predict\_filter\_noise\_dir**,**image\_name**),** image**)**

duration**=**datetime**.**datetime**.**now**()-**start\_time

**print(**'{}/{}, {} done! ------- Duration (s): {}'**.**format**(**ii**,**icount**,**image\_name**,** duration**.**seconds**)** **)**

**附录10：Dice系数预测绝缘子串珠分割 Dice.py**

**import** cv2**,**os

**import** numpy **as** np

**from** PIL **import** Image

**from** matplotlib **import** pyplot **as** plt

image\_anno\_dir**=**r"dataset\train\_anno"

image\_anno\_predict\_dir**=**r"dataset\val\_predict\_filter\_noise"

image\_anno\_set**=**os**.**listdir**(**image\_anno\_dir**)**

image\_anno\_predict\_set**=**os**.**listdir**(**image\_anno\_predict\_dir**)**

dice\_list**=[]**

**for** image\_name **in** image\_anno\_set**:**

image\_anno\_name**=**os**.**path**.**join**(**image\_anno\_dir**,**image\_name**)**

image\_anno\_predict\_name**=**os**.**path**.**join**(**image\_anno\_predict\_dir**,**image\_name**)**

image\_anno**=**cv2**.**imread**(**image\_anno\_name**)**

image\_anno\_predict**=**cv2**.**imread**(**image\_anno\_predict\_name**)**

image\_anno**=**cv2**.**resize**(**image\_anno**,(**473**,**473**))**

image\_anno\_predict**=**cv2**.**resize**(**image\_anno\_predict**,(**473**,**473**))**

image\_anno**=**image\_anno**[:,:,**0**]**

image\_anno\_predict**=**image\_anno\_predict**[:,:,**0**]**

image\_anno**[**image\_anno**>**0**]=**1

image\_anno\_predict**[**image\_anno\_predict**>**0**]=**1

# image\_anno=image\_anno/255

# image\_anno\_predict=image\_anno\_predict/255

union **=** image\_anno **\*** image\_anno\_predict

dice **=** 2**\***np**.sum(**union**)/(**np**.sum(**image\_anno**)+**np**.sum(**image\_anno\_predict**))**

dice\_list**.**append**(**dice**)** #两个样本的dice系数

**print(**image\_name**,**':'**,**dice**)**

dice**=**np**.**mean**(**dice\_list**)** #平均dice系数

**print(**'平均dice系数:'**,**dice**)**

**附录11：IOU评价绝缘子自爆区域 IOU.py**

**def** IOU**(**Reframe**,** GTframe**):**

# 得到第一个矩形的左上坐标及宽和高

x1 **=** Reframe**[**0**]**

y1 **=** Reframe**[**1**]**

width1 **=** Reframe**[**2**]**

height1 **=** Reframe**[**3**]**

# 得到第二个矩形的左上坐标及宽和高

x2 **=** GTframe**[**0**]**

y2 **=** GTframe**[**1**]**

width2 **=** GTframe**[**2**]**

height2 **=** GTframe**[**3**]**

# 计算重叠部分的宽和高

endx **=** **max(**x1 **+** width1**,** x2 **+** width2**)**

startx **=** **min(**x1**,** x2**)**

width **=** width1 **+** width2 **-** **(**endx **-** startx**)**

endy **=** **max(**y1 **+** height1**,** y2 **+** height2**)**

starty **=** **min(**y1**,** y2**)**

height **=** height1 **+** height2 **-** **(**endy **-** starty**)**

# 如果重叠部分为负, 即不重叠

**if** width **<=** 0 **or** height **<=** 0**:**

ratio **=** 0

**else:**

Area **=** width **\*** height

Area1 **=** width1 **\*** height1

Area2 **=** width2 **\*** height2

ratio **=** Area **\*** 1.0 **/** **(**Area1 **+** Area2 **-** Area**)**

**return** ratio

1. 当Dice为1的时候最好：即Dice越接近1则该模型越好。反之，当Dice为0的时候最差：即Dice越接近0，则该模型越差。 [↑](#footnote-ref-0)