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 128×2048。

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 7360×4912 的原图像及其掩模。

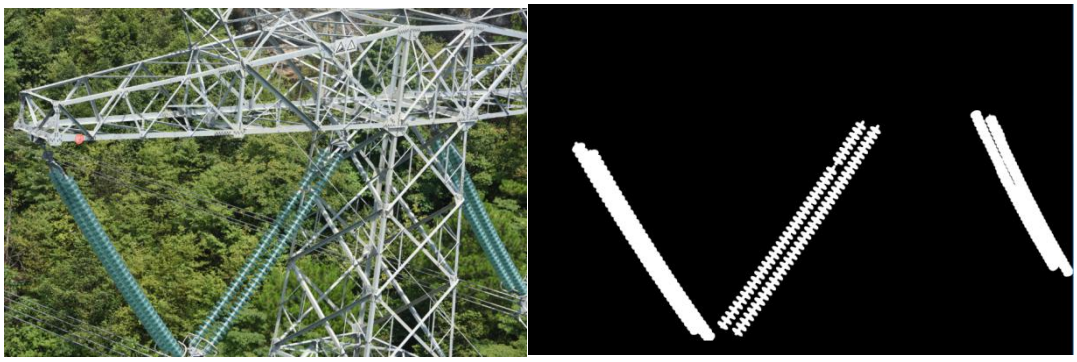


图 1 航拍高分辨率的绝缘子图像及其掩模

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。

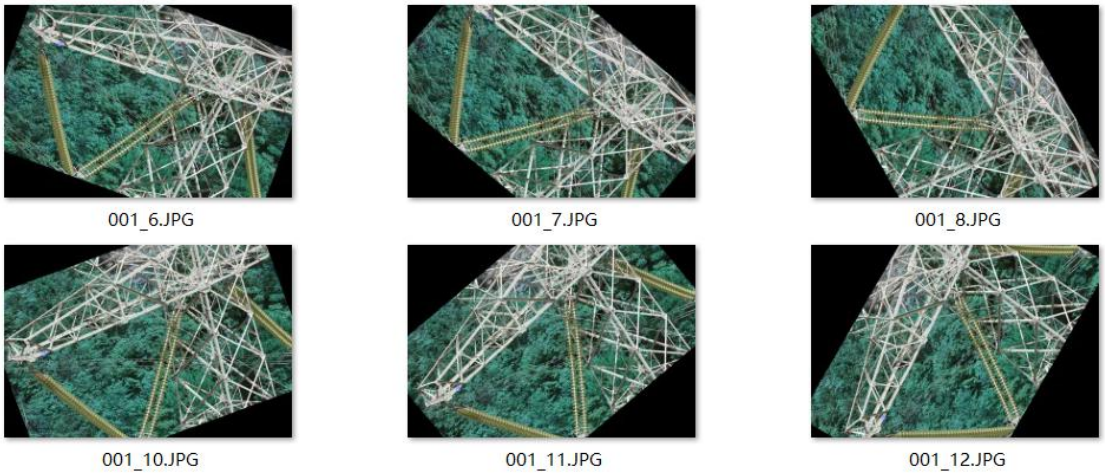


图 2 原图旋转后得到的图像

（4）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。

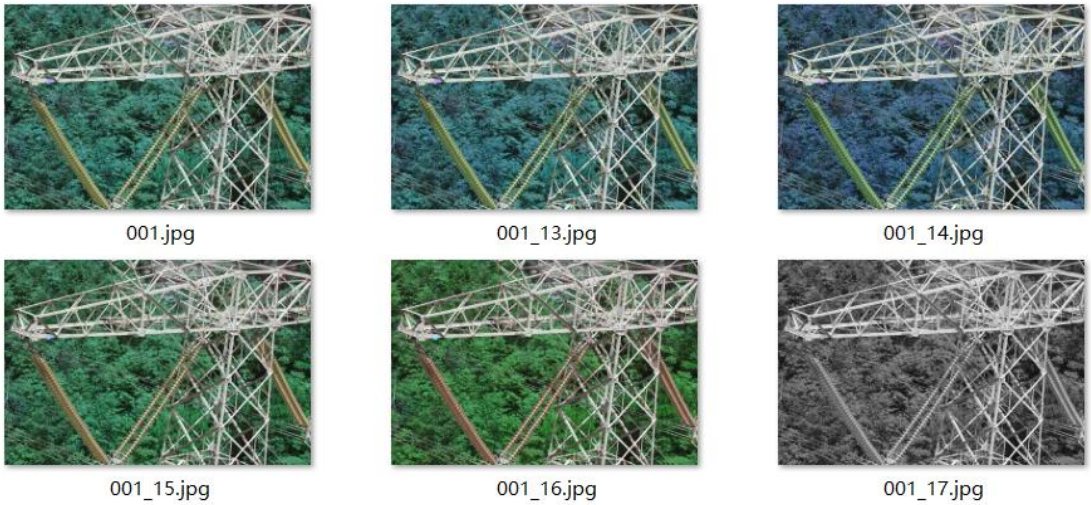


图 3 色彩变换得到的图像

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。

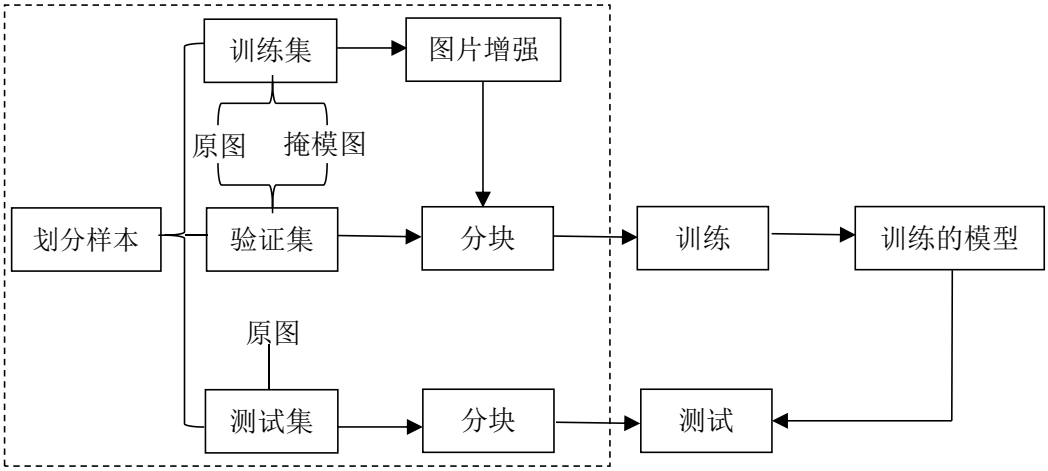


图 5 图像数据预处理流程图

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。

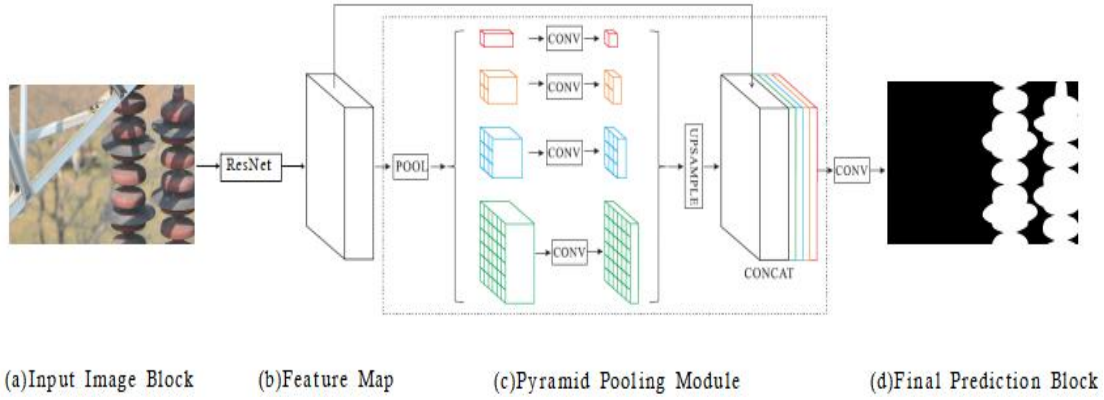


图 6 PSPNet 的网络结构

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：

$$L_{CE} = -\frac{1}{N}\sum_{i=1}^{N} < \boldsymbol{y}_i, \log \hat{\boldsymbol{y}}_i >$$

其中$\boldsymbol{y}_i$是像素点$\boldsymbol{x}_i$的真实类别标签的热编码向量，$\hat{\boldsymbol{y}}_i$是样本$\boldsymbol{x}_i$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$\boldsymbol{x}_i$绝大部分都是背景，所以L_{CE}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：

$$L_{proposed} = -\frac{1}{N_1}\sum_{\boldsymbol{x}_i \in S_1} < \boldsymbol{y}_i, \log \hat{\boldsymbol{y}}_i > + \lambda L_{CE}$$

其中S_1是指绝缘子区域的坐标集合。也就是说$L_{proposed}$对绝缘子区域的识别精度进行了加权，当$\lambda \to 0$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。

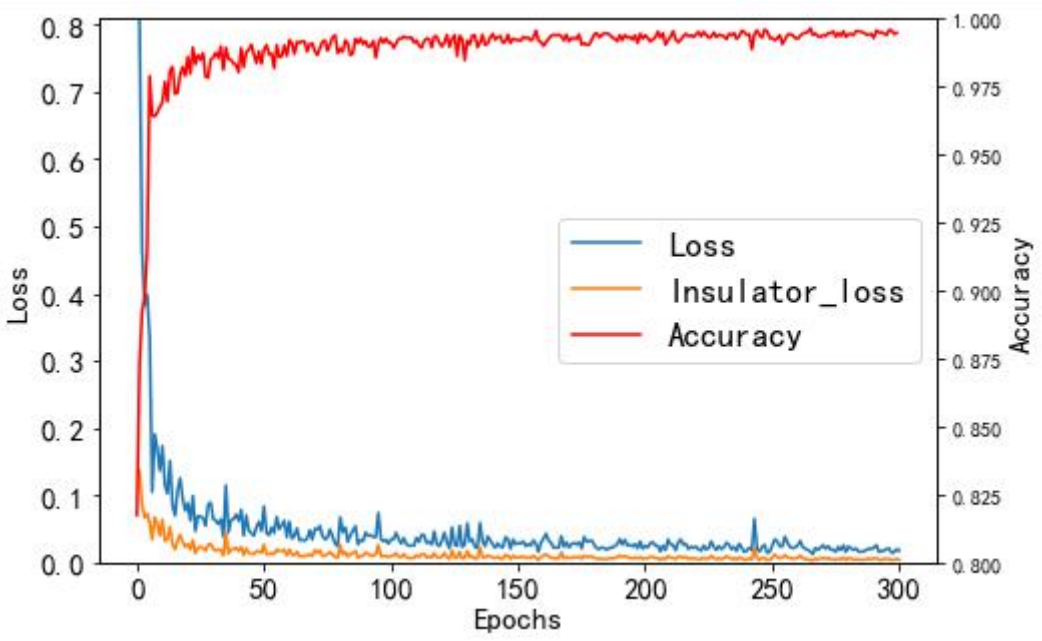


图 8 PSPNet 网络的训练过程参数的变化情况

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。

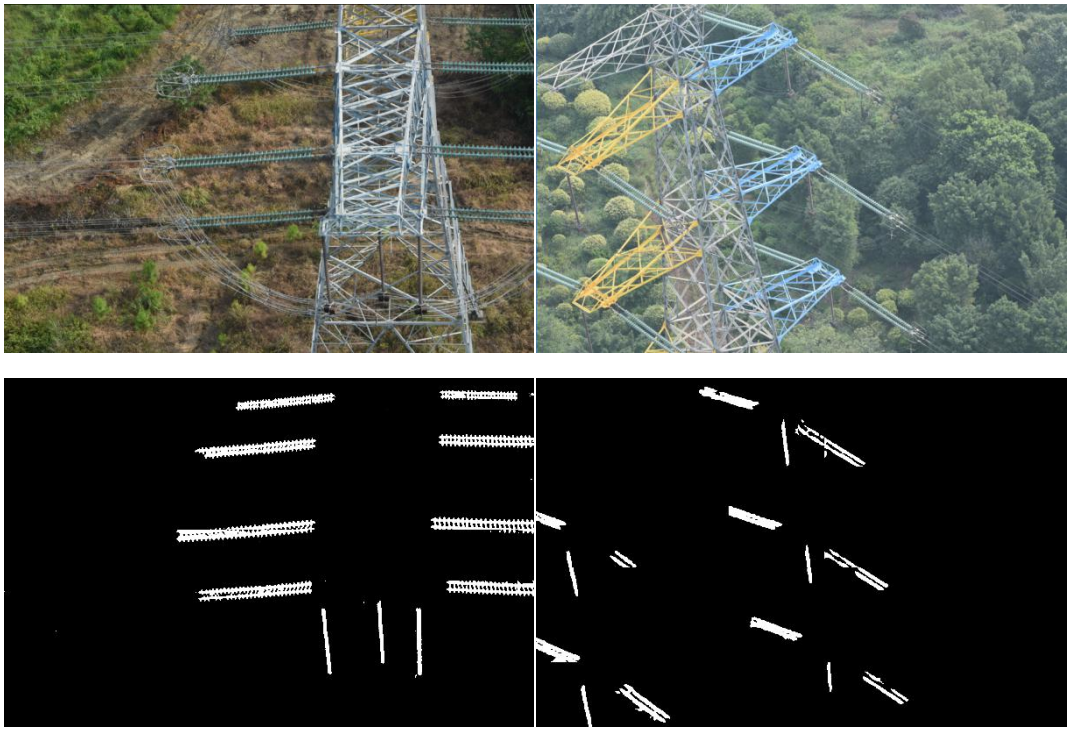


图 9 测试样本（上）及其网络模型的预测掩模图（下）

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。

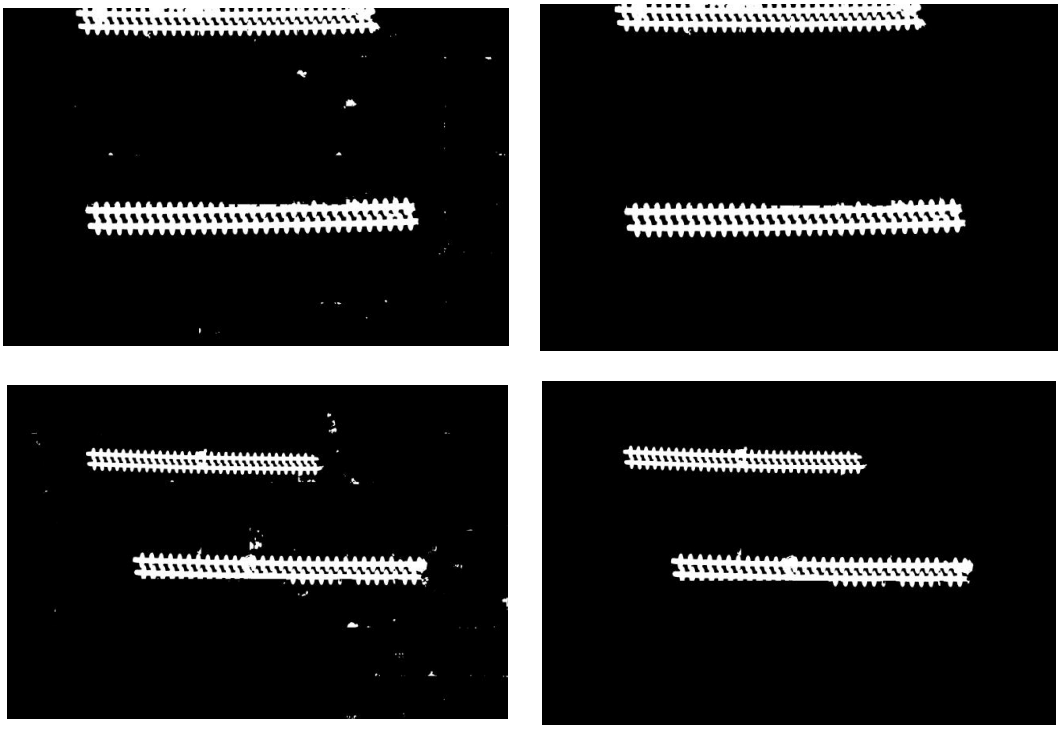


图 10 原预测掩模图（左）以及过滤噪音后的掩模图（右）

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] 系数进行评价，即评价指标：

$$\mathrm{Dice}(A,B)=\frac{2|A\cap B|}{|A|+|B|}$$

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 128×2048。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。

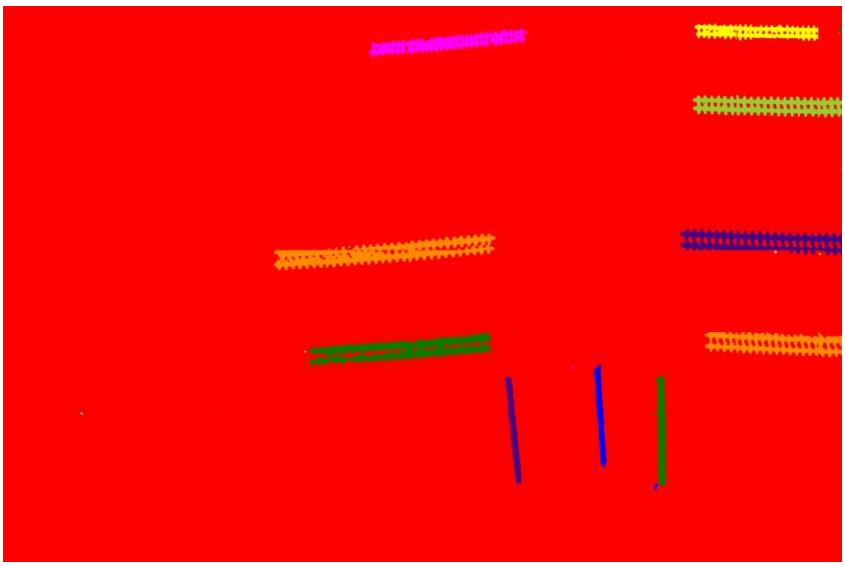


图 11 用种子填充算法找到的连通区域

（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 $S=\{\boldsymbol{x}_i\in R\times R|i=1,\cdots,n\}$，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：

[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。

$$\Sigma = \sum_{i=1}^{n} (\boldsymbol{x}_i - \boldsymbol{m})(\boldsymbol{x}_i - \boldsymbol{m})^T$$

其中 $\boldsymbol{m}$ 是均值点的坐标。我们求Σ的两个二维的特征向量$\boldsymbol{\mu}_1$和$\boldsymbol{\mu}_2$，满足：

$$\Sigma\boldsymbol{\mu} = \lambda\boldsymbol{\mu}$$

令$\boldsymbol{\mu}_1$是最大特征根对应的特征向量，则就是坐标集合 S 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 S，则$\boldsymbol{\mu}_1$就是图中的 PCA1 的方向，$\boldsymbol{\mu}_2$就是 PCA2 的方向。

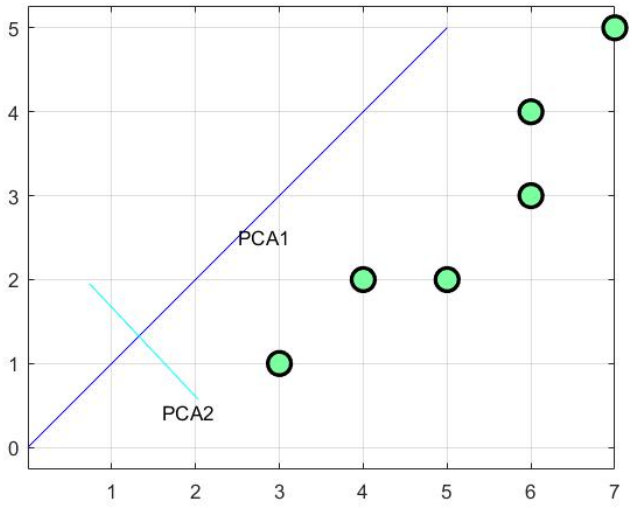


图 12 坐标点的主成分分析示意图

（3）规范化绝缘子图像。接下来我们把坐标集合 S 中的点在两个特征向量$\boldsymbol{\mu}_1$和$\boldsymbol{\mu}_2$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 l，PCA2 上的坐标之差的最大值作为高度 h。因为绝缘子串是长条的，**如果$\frac{l}{h} < 3$，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

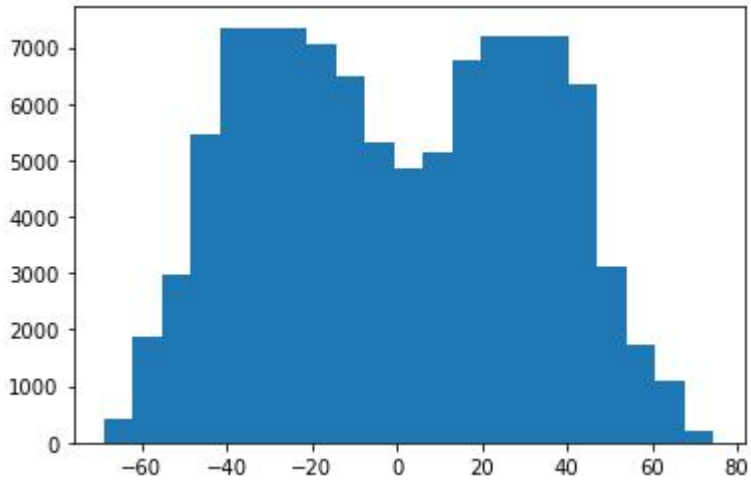


图 14 并排两个绝缘子的坐标在第 2 个主成分上的投影点的直方图

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。

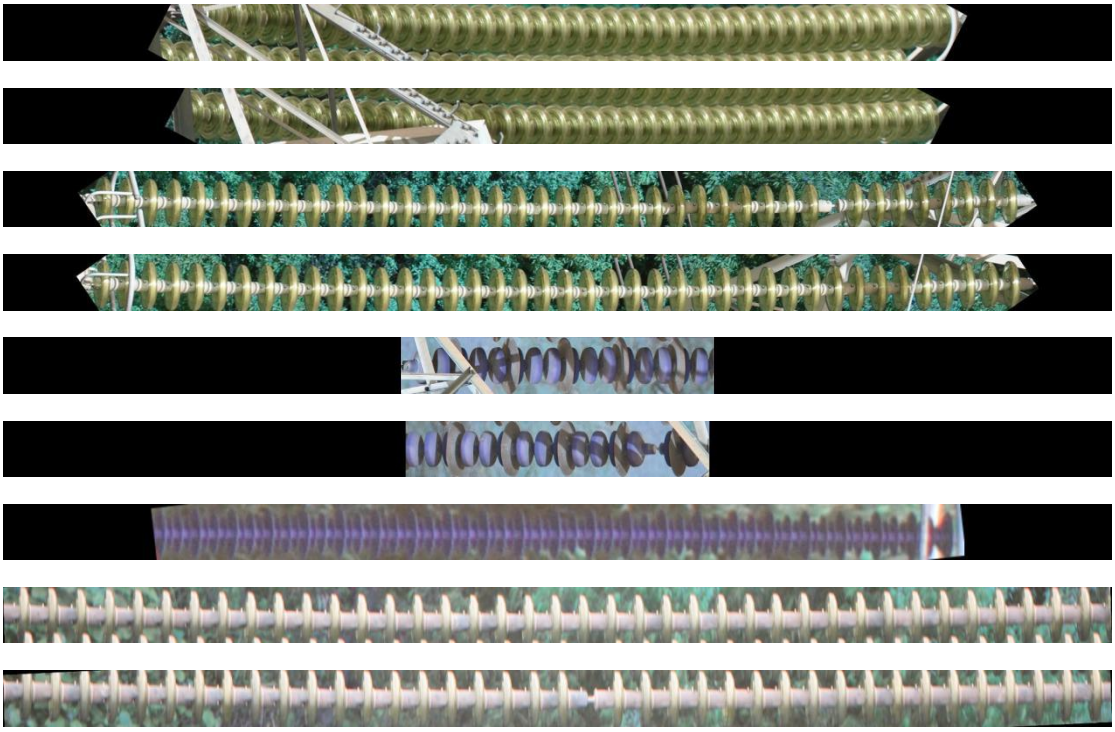


图 15 一些绝缘子规范化后的图像

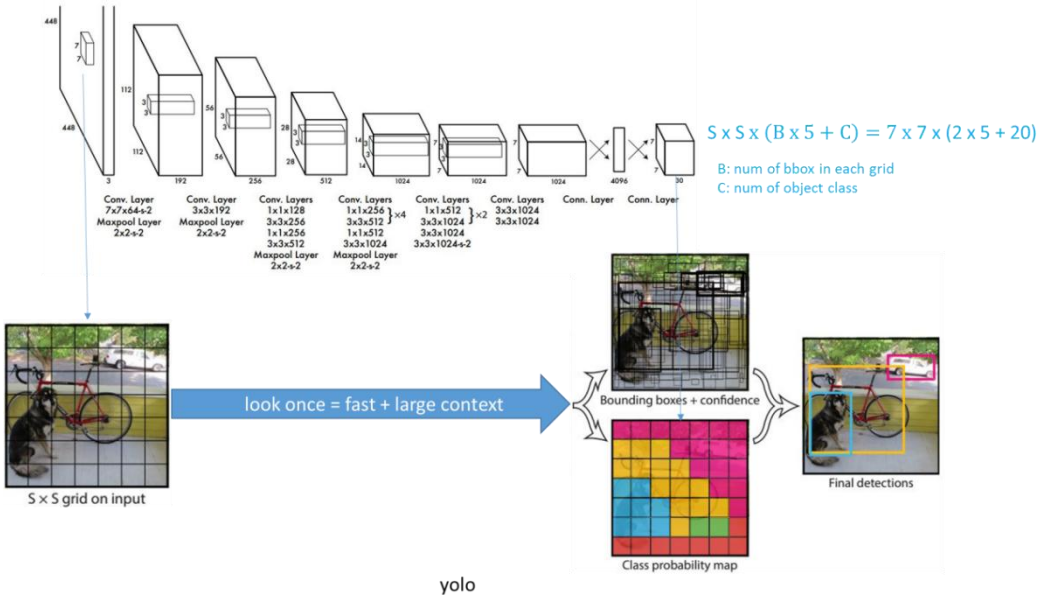
5 自爆点定位

（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 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。



yolo

	Type	Filters	Size		Output	
	Convolutional	32	3	3	256	256
	Convolutional	64	3	3 / 2	128	128
1	Convolutional	32	1	1		
	Convolutional	64	3	3		
	Residual				128	128
	Convolutional	128	3	3 / 2	64	64
2	Convolutional	64	1	1		
	Convolutional	128	3	3		
	Residual				64	64
	Convolutional	256	3	3 / 2	32	32
8	Convolutional	128	1	1		
	Convolutional	256	3	3		
	Residual				32	32
	Convolutional	512	3	3 / 2	16	16
8	Convolutional	256	1	1		
	Convolutional	512	3	3		
	Residual				16	16
	Convolutional	1024	3	3 / 2	8	8
4	Convolutional	512	1	1		
	Convolutional	1024	3	3		
	Residual				8	8
	Avgpool		Global			
	Connected		1000			
	Softmax					

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。

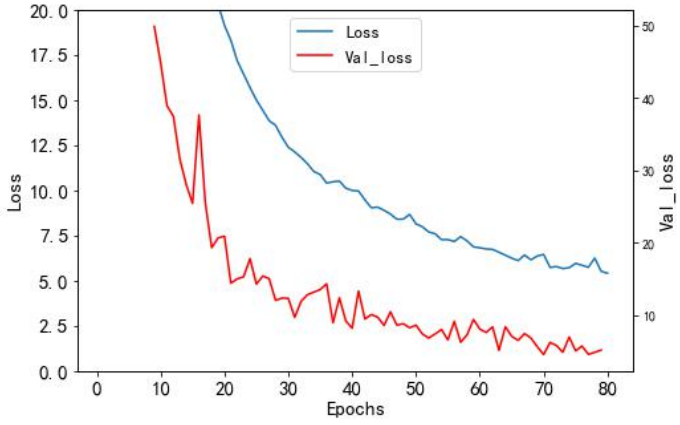


图 17 YOLO 模型的训练过程

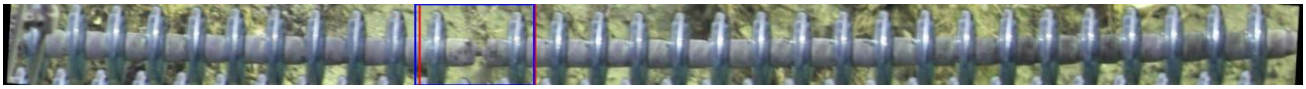
由上表数据可得训练误差变化图如图 17，由图可以看出该模型对数据处理收敛的非常快，最终 loss 与 val_loss 达到了比较小的值，效果很好。

（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。

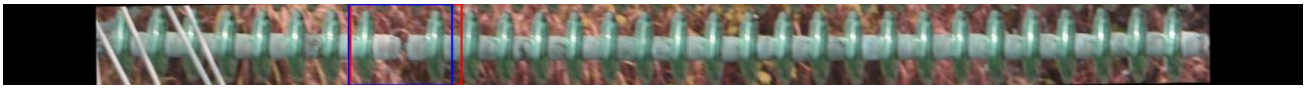
坐标(1459.2075, 32.697155)(1618.2595,119.41858)

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

（5）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。

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：

$$\mathrm{IOU} = \frac{area(C) \cap area(G)}{area(C) \cup area(G)}$$

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

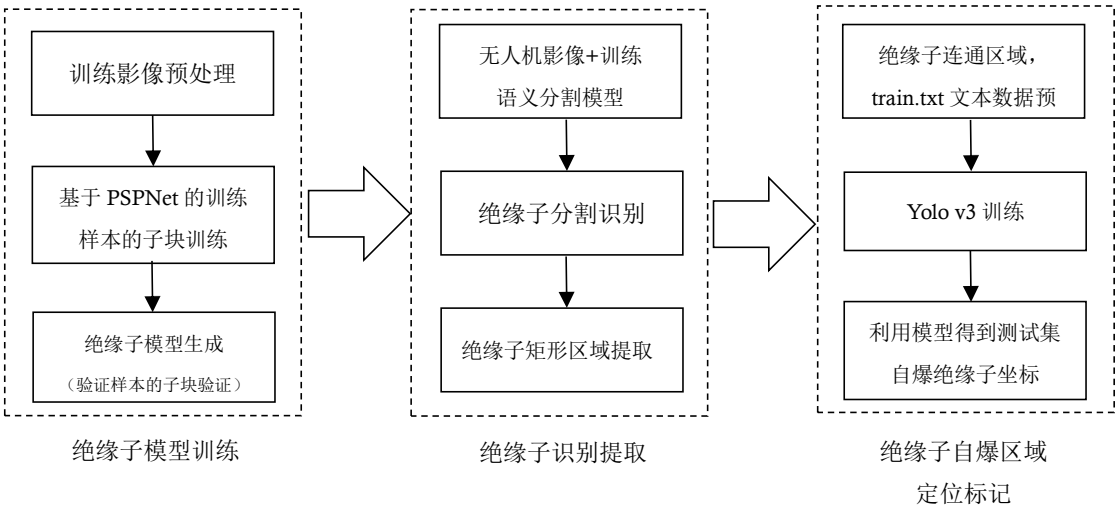


图 20 绝缘子识别及缺陷检测算法的流程图

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
```
