基于YOLOv4的销钉缺陷检测

引言

本次实验是对输电线路的图像，通过使用神经网络来识别图像里的销钉是否有缺陷。通过使用YOLOv4神经网络对输电线路的图片进行训练，以达到电脑识别图片里的销钉是否正常或者有缺失。

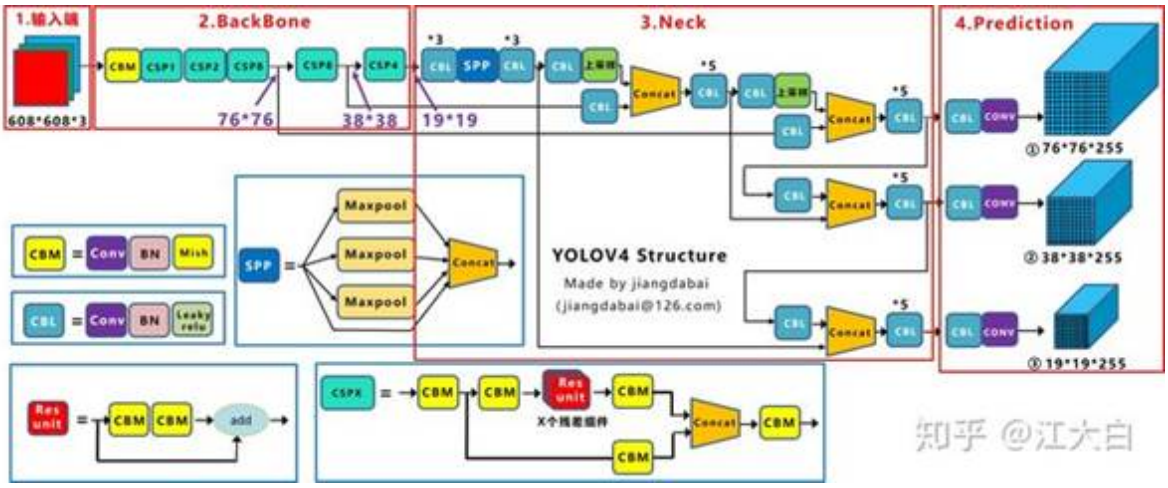
电力金具上销钉松动、缺失等常见缺陷严重影响着电力系统的稳定运行，因此有必要研究一种准确而又高效的缺陷智能识别方法。主要技术难点如下：1，即使是富有专业经验的人员进行标注，效率也是十分低下的，更深的网络结构在训练时需要大量标记数据才能保证检测精度高；2，销钉在图片中所占比例是十分微小的并且单张图片中可能存在多个销钉，致使人工标注经常出现漏检的情况；3，由于无人机拍摄角度的原因，松动的销钉和正常情况下的销钉是十分相似的，这使得容易出现错标的情形。4，销钉具有多种形状，且被安装在不同的位置。5.数据集中缺少销钉的图像很少，导致训练集中标准样本和缺陷样本不平衡，导致分类效果不足；6.销钉太小以至于很难将目标与背景图像区分开来。7. 图片采集过程中，由于抖动和光强导致图像模糊。

数据集

由于图片分辨率太大，销钉是否有缺失图片特征不明显，所以将原始图片裁剪后再训练网络。在本次实验中，首先制作了1000张图片作为训练集，检测的类比分为缺失和正常两类，分别如下图所示。

实验内容

由于输电线路数据图片背景信息负责，分辨率大，销钉是否缺失特征不明显，图片分类网络效果不好，所以采取先目标检测网络。本次实验选取YOLOv4 Darknet版，Pytorch版本进行实验，基础知识点分析见参考。网络结构图如下所示。



所需工具：

LabelImg：图像注释工具

WinSCP: 向开发板设备传输文件数据

Termius：SSH客户端工具

Pycharm：开发平台

Pytorch框架训练

下载实验代码，Pytorch训练步骤如下：

1. 训练前将**LabelImg**工具标注好的标签文件放在VOCdevkit文件夹下的VOC2007文件夹下的**Annotation**中。
2. 训练前将图片文件放在VOCdevkit文件夹下的VOC2007文件夹下的JPEGImages中。
3. 在训练前利用voc2yolo3.py文件生成对应的**txt**文档。
4. 再运行根目录下的voc_annotation.py，生成2007_train.txt，每一行对应其**图片位置**及其**真实框的位置**。对这个代码做修改可以生成darknet所需的train.txt。**注意**Darknet训练时图片路径最好**不能有汉字**，现有的截图图片名包含截图两个汉字，需要将图片名转换一下。
5. 运行train.py即可开始训练。

利用pytorch搭建网络结构的代码如下：

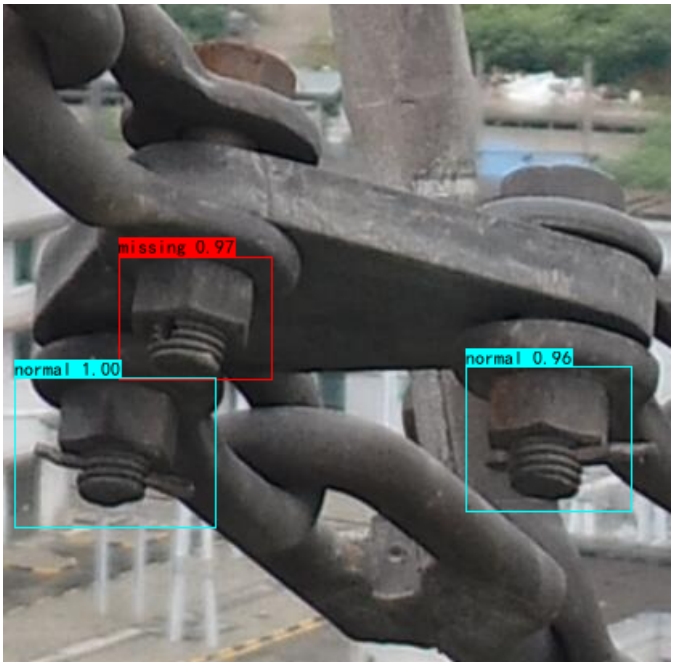
```
from collections import OrderedDict
import torch
import torch.nn as nn

from nets.CSPdarknet import darknet53

def conv2d(filter_in, filter_out, kernel_size, stride=1):
    pad = (kernel_size - 1) // 2 if kernel_size else 0
```

```
    return nn.Sequential(OrderedDict([
        ("conv", nn.Conv2d(filter_in, filter_out, kernel_size=kernel_size,
stride=stride, padding=pad, bias=False)),
        ("bn", nn.BatchNorm2d(filter_out)),
        ("relu", nn.LeakyReLU(0.1)),
    ]))

class SpatialPyramidPooling(nn.Module):
    def __init__(self, pool_sizes=[5, 9, 13]):
        super(SpatialPyramidPooling, self).__init__()

        self.maxpools = nn.ModuleList([nn.MaxPool2d(pool_size, 1, pool_size//2)
for pool_size in pool_sizes])

    def forward(self, x):
        features = [maxpool(x) for maxpool in self.maxpools[::-1]]
        features = torch.cat(features + [x], dim=1)

        return features
class Upsample(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(Upsample, self).__init__()

        self.upsample = nn.Sequential(
            conv2d(in_channels, out_channels, 1),
            nn.Upsample(scale_factor=2, mode='nearest')
        )

    def forward(self, x,):
        x = self.upsample(x)
        return x
def make_three_conv(filters_list, in_filters):
    m = nn.Sequential(
        conv2d(in_filters, filters_list[0], 1),
        conv2d(filters_list[0], filters_list[1], 3),
        conv2d(filters_list[1], filters_list[0], 1),
    )
    return m
def make_five_conv(filters_list, in_filters):
    m = nn.Sequential(
        conv2d(in_filters, filters_list[0], 1),
        conv2d(filters_list[0], filters_list[1], 3),
        conv2d(filters_list[1], filters_list[0], 1),
        conv2d(filters_list[0], filters_list[1], 3),
        conv2d(filters_list[1], filters_list[0], 1),
    )
    return m
def yolo_head(filters_list, in_filters):
    m = nn.Sequential(
        conv2d(in_filters, filters_list[0], 3),
        nn.Conv2d(filters_list[0], filters_list[1], 1),
    )
    return m
class YoloBody(nn.Module):
    def __init__(self, num_anchors, num_classes):
        super(YoloBody, self).__init__()
        self.backbone = darknet53(None)
```

```
        self.conv1 = make_three_conv([512,1024],1024)
        self.SPP = SpatialPyramidPooling()
        self.conv2 = make_three_conv([512,1024],2048)

        self.upsample1 = Upsample(512,256)
        self.conv_for_P4 = conv2d(512,256,1)
        self.make_five_conv1 = make_five_conv([256, 512],512)

        self.upsample2 = Upsample(256,128)
        self.conv_for_P3 = conv2d(256,128,1)
        self.make_five_conv2 = make_five_conv([128, 256],256)

        final_out_filter2 = num_anchors * (5 + num_classes)
        self.yolo_head3 = yolo_head([256, final_out_filter2],128)

        self.down_sample1 = conv2d(128,256,3,stride=2)
        self.make_five_conv3 = make_five_conv([256, 512],512)

        final_out_filter1 =  num_anchors * (5 + num_classes)
        self.yolo_head2 = yolo_head([512, final_out_filter1],256)

        self.down_sample2 = conv2d(256,512,3,stride=2)
        self.make_five_conv4 = make_five_conv([512, 1024],1024)

        final_out_filter0 =  num_anchors * (5 + num_classes)
        self.yolo_head1 = yolo_head([1024, final_out_filter0],512)


    def forward(self, x):
        x2, x1, x0 = self.backbone(x)

        P5 = self.conv1(x0)
        P5 = self.SPP(P5)
        P5 = self.conv2(P5)

        P5_upsample = self.upsample1(P5)
        P4 = self.conv_for_P4(x1)
        P4 = torch.cat([P4,P5_upsample],axis=1)
        P4 = self.make_five_conv1(P4)

        P4_upsample = self.upsample2(P4)
        P3 = self.conv_for_P3(x2)
        P3 = torch.cat([P3,P4_upsample],axis=1)
        P3 = self.make_five_conv2(P3)

        P3_downsample = self.down_sample1(P3)
        P4 = torch.cat([P3_downsample,P4],axis=1)
        P4 = self.make_five_conv3(P4)

        P4_downsample = self.down_sample2(P4)
        P5 = torch.cat([P4_downsample,P5],axis=1)
        P5 = self.make_five_conv4(P5)
        out2 = self.yolo_head3(P3)
        out1 = self.yolo_head2(P4)
        out0 = self.yolo_head1(P5)
        return out0, out1, out2
```

实验结果

YOLOv3的loss在3.2左右，YOLOv4的loss在0.76。运行predict.py进行图片检测, PC上模型测试效果如下：



Darknet框架训练

在**虚拟机**上实验，用户**密码为esr@**。先将图片等所需的文件上传到虚拟机上。实验步骤参考，步骤如下：

1. clone项目源代码，下载预训练的权重文件：yolov4.conv.137
2. 修改根目录下cfg文件夹里面的yolov4.cfg如下：

- 修改合适大小的 batch=32
- 修改subdivisions=16
- 将行 max_batches 更改为 (classes*2000，但不少于训练图像的数量且不少于6000)，这里设置为6000。将steps 更改为 max_batches的 80% 和 90% 如：steps=4800,5400
- 设置网络大小width=416，height=416或 32 的任何值倍数。这里为416
- 将classes=80 更改classes=2，三处修改
- filters=255更改为 filters=21 （(classes + 5)x3），三处修改

3. 修改根目录下的makefile文件，配置好后再编译make

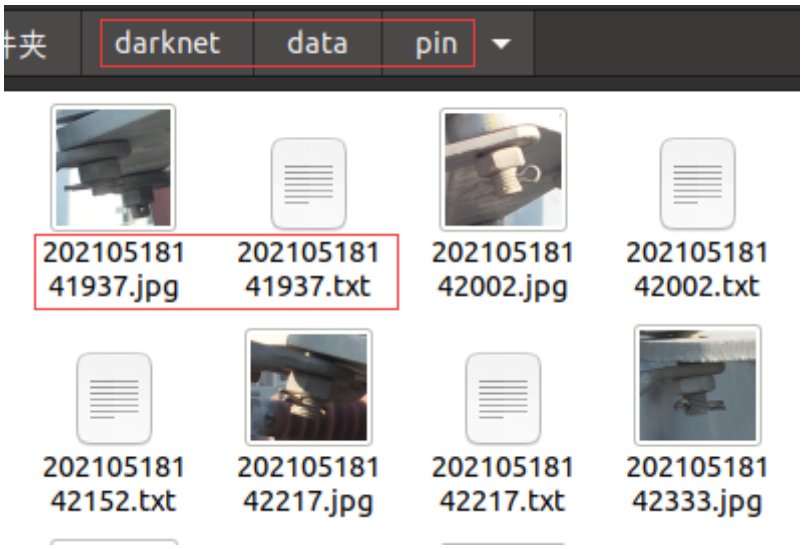
GPU=0
CUDNN=0
CUDNN_HALF=0
OPENCV=0

4. 在目录/home/esr/darknet/data/中创建具有类别名称的文件pin.names，内容如下

```
missing
normal
```

5. 在目录/home/esr/darknet/data/中创建文件pin.data，所需的文件也放入相应目录中。train.txt里包含了训练图片的路径。内容如下。训练图片路径内容示例如下图，

```
classes= 2
train  = /home/esr/darknet/data/train.txt
valid  = /home/esr/darknet/data/valid.txt
names = /home/esr/darknet/data/pin.names
backup = /home/esr/darknet/backup/
```



6. 将图像文件放入目录中/home/esr/darknet/data/pin/中。训练图片和标签txt文件需放在同一文件夹下，对于每个txt文档内容示例如下：

```
<object-class> <x_center> <y_center> <width> <height>
```

标签xml文档转换成yolo所需的格式代码参考如下：

```
import os
import xml.etree.ElementTree as ET
from os import getcwd

sets = [('2007', 'train'), ('2007', 'val'), ('2007', 'test')]

classes = ["missing", "normal"]

def convert(size, box):
    dw = 1. / size[0]
    dh = 1. / size[1]
    x = (box[0] + box[1]) / 2.0
    y = (box[2] + box[3]) / 2.0
    w = box[1] - box[0]
    h = box[3] - box[2]
    x = x * dw
    w = w * dw
    y = y * dh
    h = h * dh
    return (x, y, w, h)

def convert_annotation(year, image_id, list_file):
    in_file = open('VOCdevkit/VOC%s/Annotations/%s.xml' % (year, image_id),
encoding='utf-8')
    out_file = open(wd + '/labels/%s.txt' % (image_id), 'w')
    tree = ET.parse(in_file)
    root = tree.getroot()
```

```
    size = root.find('size')
    w = int(size.find('width').text)
    h = int(size.find('height').text)
    for obj in root.iter('object'):
        difficult = 0
        if obj.find('difficult') != None:
            difficult = obj.find('difficult').text

        cls = obj.find('name').text
        if cls not in classes or int(difficult) == 1:
            continue
        cls_id = classes.index(cls)
        xmlbox = obj.find('bndbox')
        b = (int(xmlbox.find('xmin').text), int(xmlbox.find('xmax').text),
int(xmlbox.find('ymin').text), int(xmlbox.find('ymax').text))
        bb = convert((w, h), b)
        out_file.write(str(cls_id) + " " + " ".join([str(a) for a in bb]) +
'\n')

if __name__ == '__main__':
    wd = getcwd()
    wd = wd.replace('\\', '/')
    for year, image_set in sets:
        if not os.path.exists(wd + '/labels/'):
            os.makedirs(wd + '/labels/')
        image_ids = open('VOCdevkit/VOC%s/ImageSets/Main/%s.txt' % (year,
image_set)).read().strip().split()
        list_file = open('%s_%s.txt' % (year, image_set), 'w')
        for image_id in image_ids:
            list_file.write(wd + '/JPEGImages/%s.jpg\n' % image_id)
            convert_annotation(year, image_id, image_id)
        list_file.close()
```

7. 使用命令行开始训练： `./darknet detector train data/pin.data cfg/yolov4.cfg yolov4.conv.137 -map`

在RK3399开发板上实验

Pytorch模型序列化

只凭一个网络参数， RKNN Toolkit 无法构建相应的网络。所以首先需要将网络结构和权重固化成一个开发板能使用的torch.jit.trace 保存的模型。通常的做法定义一个net，然后再net.load_state_dict 加载pth权重，最后再用torch.jit.trace将网络结构和权重固化成一个pt文件 ，然后再用rknn.load_pytorch对这个pt文件进行转换。转换代码如下

```
import numpy as np
import torch

from nets.yolo3 import YoloBody
model = YoloBody(3,2)
model_path = "logs/Epoch99-Total_Loss2.8750-Val_Loss3.2312.pth"
print('Loading weights into state dict...')
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model_dict = model.state_dict()
pretrained_dict = torch.load(model_path, map_location=device)
```

```
pretrained_dict = {k: v for k, v in pretrained_dict.items() if
np.shape(model_dict[k]) == np.shape(v)}
model_dict.update(pretrained_dict)
model.load_state_dict(model_dict)
print('Finished!')

model.eval()
example = torch.rand(1, 3, 416, 416)
with torch.no_grad():
    traced_script_module = torch.jit.trace(model, example)

traced_script_module.save('./yolov3-jit.pt')
```

量化模型

为了确保量化后精度与原来模型能对,需要：1.使用多张图进行量化，确保量化精度稳定。2.设置 mean_values/std_values 参数，确保其和训练模型时使用的参数相同。由于模型过大，开发板内存不足，需要**在PC上**安装好rknn.api后进行模型量化。rknn-toolkit-v1.6.1增加了支持pytorch的算子。

量化代码如下：

```
if __name__ == '__main__':
    MODEL_PATH = "yolov3-jit.pt"
    RKNN_MODEL_PATH = './yolo3.rknn'
    im_file = './4.jpg'
    DATASET = './dataset.txt'
    input_size_list = [[3, 416, 416]]
    # Create RKNN object
    rknn = RKNN()

    NEED_BUILD_MODEL = True

    if NEED_BUILD_MODEL:
        # pre-process config
        print('--> Config model')
        rknn.config(batch_size=1,epochs=1000,mean_values=[[0, 0, 0]],
std_values=[[255, 255, 255]], reorder_channel='0 1 2')
        print('done')

        # Load Pytorch model
        print('--> Loading model')
        ret = rknn.load_pytorch(model=MODEL_PATH,
input_size_list=input_size_list)
        if ret != 0:
            print('Load Pytorch model failed!')
            exit(ret)
        print('done')

        # Build model
        print('--> Building model')
        ret = rknn.build(do_quantization=True)
        if ret != 0:
            print('Build model failed!')
            exit(ret)
        print('done')
        # Export RKNN model
        print('--> Export RKNN model')
```

```
        ret = rknn.export_rknn('./yolo3.rknn')
        if ret != 0:
            print('Export yolo4.rknn failed!')
            exit(ret)
        print('done')
    else:
        # Direct load rknn model
        print('Loading RKNN model')
        ret = rknn.load_rknn(RKNN_MODEL_PATH)
        if ret != 0:
            print('Load RKNN model failed.')
            exit(ret)
        print('done')
```

开发板上测试

首先参考官方文档链接准备实验环境，下载所需文档、安装包和驱动程序等。在PC和开发板上都需要安装所需Python依赖和RKNN-Toolkit，开发板的用户密码都是**toybrick**。实验步骤如下：

1. 环境准备

   将 RK3399开发板通过 USB 连接到 PC 上。第一次使用开发板时需要安装相应的驱动，安装方式如下：进入SDK 包 platform-tools/drivers_installer/windows-x86_64 目录，以管理员身份运行 zadig-2.4.exe 程序安装计算棒的驱动。

2. 安装 RKNN-Toolkit

   安装 RKNN-Toolkit 前需要确保系统里已经安装有 Python3.6。获取 RKNN-Toolkit SDK 包链接，然后安装 Python 依赖：
   pip install tensorflow==1.14.0
   pip install torch==1.6.0+cpu torchvision==0.7.0+cpu
   安装RKNN-Toolkit：pip install rknn_toolkit-1.6.1-cp36-cp36m-win_amd64.whl

3. 运行示例

   将实验所需的代码和数据通过WinSCP上传到开发板上，然后通过Termius客户端，进入到相应目录，执行 test.py 的脚本即可测试：

本实验**测试Pytorch模型**的代码如下，测试Darknet模型的代码参考**rknn-toolkit-v1.6.1/examples/darknet/yolov3/**下的test.py文件。

```
import numpy as np
import cv2
from rknn.api import RKNN

GRID0 = 13
GRID1 = 26
GRID2 = 52
LISTSIZE = 7
SPAN = 3
NUM_CLS = 2
OBJ_THRESH = 0.55
NMS_THRESH = 0.4

CLASSES = ("missing", "normal")

def sigmoid(x):
    return 1 / (1 + np.exp(-x))
```

```
def process(input, mask, anchors):

    anchors = [anchors[i] for i in mask]
    grid_h, grid_w = map(int, input.shape[0:2])

    box_confidence = sigmoid(input[..., 4])
    box_confidence = np.expand_dims(box_confidence, axis=-1)

    box_class_probs = sigmoid(input[..., 5:])

    box_xy = sigmoid(input[..., :2])
    box_wh = np.exp(input[..., 2:4])
    box_wh = box_wh * anchors

    col = np.tile(np.arange(0, grid_w), grid_w).reshape(-1, grid_w)
    row = np.tile(np.arange(0, grid_h).reshape(-1, 1), grid_h)

    col = col.reshape(grid_h, grid_w, 1, 1).repeat(3, axis=-2)
    row = row.reshape(grid_h, grid_w, 1, 1).repeat(3, axis=-2)
    grid = np.concatenate((col, row), axis=-1)

    box_xy += grid
    box_xy /= (grid_w, grid_h)
    box_wh /= (416, 416)
    box_xy -= (box_wh / 2.)
    box = np.concatenate((box_xy, box_wh), axis=-1)

    return box, box_confidence, box_class_probs

def filter_boxes(boxes, box_confidences, box_class_probs):

    box_scores = box_confidences * box_class_probs
    box_classes = np.argmax(box_scores, axis=-1)
    box_class_scores = np.max(box_scores, axis=-1)
    pos = np.where(box_class_scores >= OBJ_THRESH)

    boxes = boxes[pos]
    classes = box_classes[pos]
    scores = box_class_scores[pos]

    return boxes, classes, scores

def nms_boxes(boxes, scores):

    x = boxes[:, 0]
    y = boxes[:, 1]
    w = boxes[:, 2]
    h = boxes[:, 3]

    areas = w * h
    order = scores.argsort()[::-1]

    keep = []
    while order.size > 0:
        i = order[0]
        keep.append(i)

        xx1 = np.maximum(x[i], x[order[1:]])
```

```
        yy1 = np.maximum(y[i], y[order[1:]])
        xx2 = np.minimum(x[i] + w[i], x[order[1:]] + w[order[1:]])
        yy2 = np.minimum(y[i] + h[i], y[order[1:]] + h[order[1:]])

        w1 = np.maximum(0.0, xx2 - xx1 + 0.00001)
        h1 = np.maximum(0.0, yy2 - yy1 + 0.00001)
        inter = w1 * h1

        ovr = inter / (areas[i] + areas[order[1:]] - inter)
        inds = np.where(ovr <= NMS_THRESH)[0]
        order = order[inds + 1]
    keep = np.array(keep)
    return keep


def yolov3_post_process(input_data):

    masks = [[6, 7, 8], [3, 4, 5], [0, 1, 2]]
    anchors = [[12, 16], [19, 36], [40, 28], [36, 75], [76, 55],
               [72, 146], [142, 110], [192, 243], [459, 401]]
    boxes, classes, scores = [], [], []
    for input,mask in zip(input_data, masks):
        b, c, s = process(input, mask, anchors)
        b, c, s = filter_boxes(b, c, s)
        boxes.append(b)
        classes.append(c)
        scores.append(s)

    boxes = np.concatenate(boxes)
    classes = np.concatenate(classes)
    scores = np.concatenate(scores)

    nboxes, nclasses, nscores = [], [], []
    for c in set(classes):
        inds = np.where(classes == c)
        b = boxes[inds]
        c = classes[inds]
        s = scores[inds]

        keep = nms_boxes(b, s)

        nboxes.append(b[keep])
        nclasses.append(c[keep])
        nscores.append(s[keep])

    if not nclasses and not nscores:
        return None, None, None

    boxes = np.concatenate(nboxes)
    classes = np.concatenate(nclasses)
    scores = np.concatenate(nscores)

    return boxes, classes, scores

def draw(image, boxes, scores, classes,colors):

    for box, score, cl in zip(boxes, scores, classes):
        x, y, w, h = box
```

```
        print('class: {}, score: {}'.format(CLASSES[cl], score))
        print('box coordinate left,top,right,down: [{}, {}, {}, {}]'.format(x,
y, x+w, y+h))
        x *= image.shape[1]
        y *= image.shape[0]
        w *= image.shape[1]
        h *= image.shape[0]
        top = max(0, np.floor(x + 0.5).astype(int))-3
        left = max(0, np.floor(y + 0.5).astype(int))-3
        right = min(image.shape[1], np.floor(x + w + 0.5).astype(int))+3
        bottom = min(image.shape[0], np.floor(y + h + 0.5).astype(int))+3
        color=CLASSES.index(CLASSES[cl])
        cv2.rectangle(image, (top, left), (right, bottom), (0, 255, 0), 1)
        cv2.putText(image, '{0} {1:.2f}'.format(CLASSES[cl], score),
                    (top, left - 6),
                    cv2.FONT_HERSHEY_SIMPLEX,
                    0.4, colors[color], 1)
        cv2.imwrite("out.jpg", image)

if __name__ == '__main__':
    MODEL_PATH = "yolov4-jit.pt"
    RKNN_MODEL_PATH = './yolo4.rknn'
    im_file = './4.jpg'
    DATASET = './dataset.txt'
    input_size_list = [[3, 416, 416]]
    # Create RKNN object
    rknn = RKNN()

    NEED_BUILD_MODEL = False

    if NEED_BUILD_MODEL:
        # pre-process config
        print('--> Config model')
        rknn.config(batch_size=1,epochs=100,mean_values=[[123.675, 116.28,
103.53]], std_values=[[58.395, 58.395, 58.395]], reorder_channel='0 1 2')
        print('done')

        # Load Pytorch model
        print('--> Loading model')
        ret = rknn.load_pytorch(model=MODEL_PATH,
input_size_list=input_size_list)
        if ret != 0:
            print('Load Pytorch model failed!')
            exit(ret)
        print('done')

        # Build model
        print('--> Building model')
        ret = rknn.build(do_quantization=True, dataset='./dataset.txt')
        if ret != 0:
            print('Build model failed!')
            exit(ret)
        print('done')

        # Export RKNN model
        print('--> Export RKNN model')
        ret = rknn.export_rknn('./yolo4.rknn')
        if ret != 0:
```

```
            print('Export yolo4.rknn failed!')
            exit(ret)
        print('done')
    else:
        # Direct load rknn model
        print('Loading RKNN model')
        ret = rknn.load_rknn(RKNN_MODEL_PATH)
        if ret != 0:
            print('Load RKNN model failed.')
            exit(ret)
        print('done')

    # Init runtime environment
    print('--> Init runtime environment')
    ret = rknn.init_runtime()
    if ret != 0:
        print('Init runtime environment failed.')
        exit(ret)
    print('done')
    img = cv2.imread(im_file)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    size = (416, 416)
    img = cv2.resize(img, size)
    img = np.array(img, dtype=np.float32)
    images = np.transpose(img / 255.0, (2, 0, 1))
    # photo = np.array(img, dtype=np.float32) / 255.0
    # photo = np.transpose(photo, (2, 0, 1))
    # images = np.asarray([photo])
    # Inference
    print('--> Running model')
    outputs = rknn.inference(inputs=[images])

    input0_data = np.array(outputs[0])
    input1_data = np.array(outputs[1])
    input2_data = np.array(outputs[2])

    input0_data = input0_data.reshape(SPAN, LISTSIZE, GRID0, GRID0)
    input1_data = input1_data.reshape(SPAN, LISTSIZE, GRID1, GRID1)
    input2_data = input2_data.reshape(SPAN, LISTSIZE, GRID2, GRID2)

    input_data = list()
    input_data.append(np.transpose(input0_data, (2, 3, 0, 1)))
    input_data.append(np.transpose(input1_data, (2, 3, 0, 1)))
    input_data.append(np.transpose(input2_data, (2, 3, 0, 1)))
    boxes, classes, scores = yolov3_post_process(input_data)

    image = cv2.imread(im_file)
    colors=[(0, 0, 255), (0, 255, 0)]
    if boxes is not None:
        draw(image, boxes, scores, classes,colors)
```

TODO:

使用原生的Darknet 框架训练的模型，经测试不会出现Pytorch模型量化后造成的**精度误差。**

在用Darknet 框架训练的时候loss在训练几个批次之后为**nan**值，可能是数据集的某些标签有问题。可以不对原图进行裁剪，重新做好标签进行训练。需要注意的是标签数量需要足够多，质量高，否则训练效果会不好。

若使用MTCNN训练，标签制作方法不一样。

其他

[1] [1] [2] [2] 考虑到深度学习中的"单阶"检测算法RetinaNet在识别精度和实时性均要优于Faster RCNN，销钉松动类样本的严重不足以及无人机收集该类样本代价高昂，其采用人工采集的相关辅助数据对少数样本进行补充训练，结果表明添加适宜的相关辅助数据量能够明显增加检测模型对少数类的重视程度。

[3] [3] 考虑到无人机现场采集图像时由于自身的飞行和抖动使得一部分图像清晰度偏低的问题，文中构建了生成对抗网络，借助GAN来提高图像的清晰度，从而提高此类数据的识别精度。通过生成器来处理模糊图像，输出所需的清晰数据。针对GAN训练的图像数据对在现实情况下很难获得的问题,文中借助马尔可夫过程生成的轨迹矢量和子像素插值来实现模糊一清晰图像对的构建,最后通过RetinaNet实现销钉缺陷的检测。

[4] [4] 基于级联卷积神经网络的目标检测方法（改进MTCNN）。在以下四个方面对原有的MTCNN进行了改进：（1）在分解卷积核后，在部分卷积层后加入非线性多层感知器，同时去除全连接层；(2) 融合了多尺度特征图；(3) 在分类交叉熵损失函数中加入一个**角度变量**；（4）在训练过程中采用多任务学习和离线困难样本挖掘策略。。训练样本分为四类：正样本、负样本、部分正样本和难样本。正样本和负样本可以提高模型的分类损失函数，而正样本和部分正样本可以提高模型的边界回归损失函数。训练过程分为两个步骤：预训练和离线困难样本训练。整个过程如图所示。

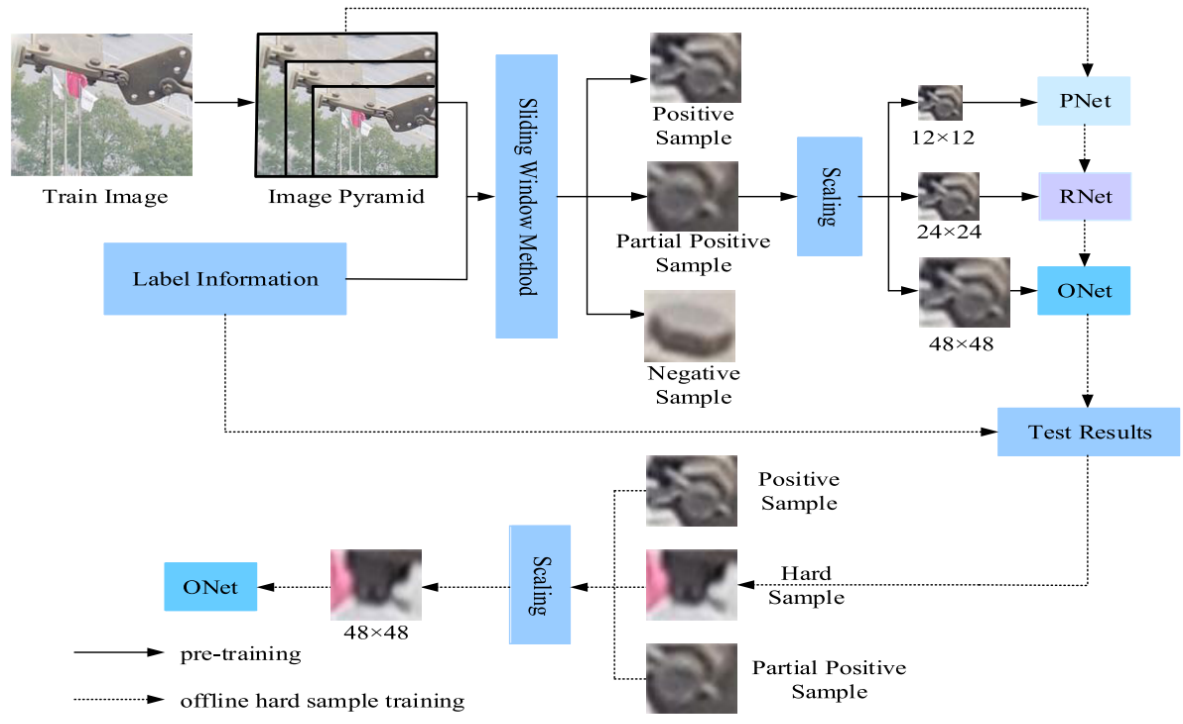


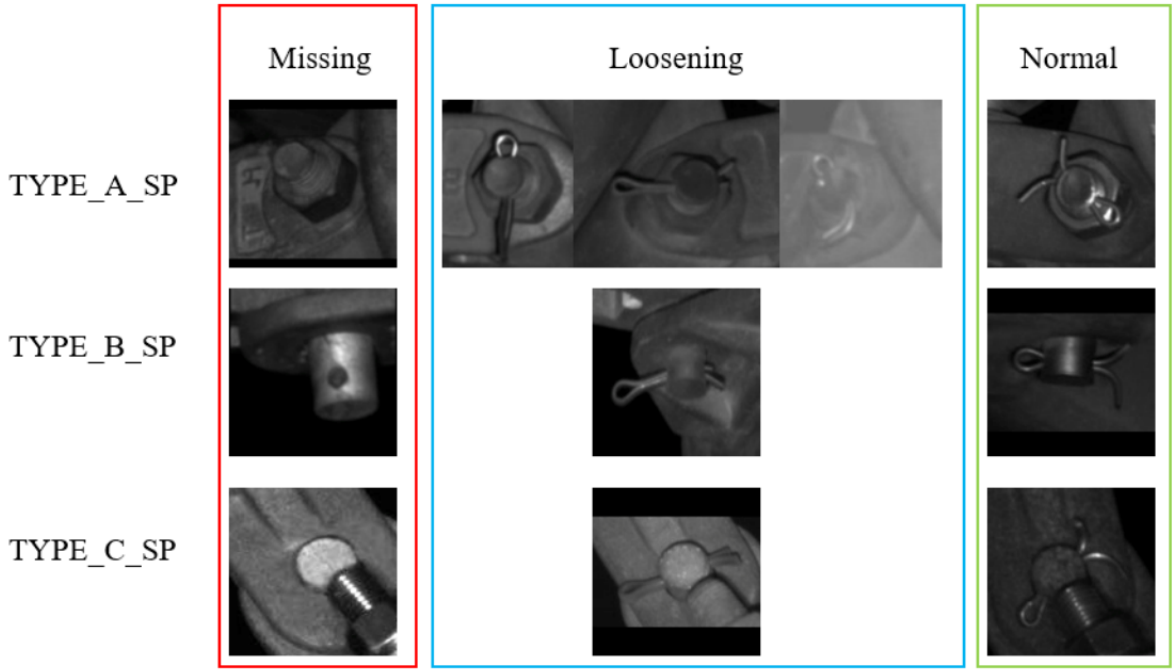
FIGURE 8. **Flowchart of training.**

数据来自三个地方：（1）高压输电线路航拍图；(2) 低压输电线路航拍图；(3) 通过互联网搜索获得的图像。包括三个地方在内，图像总数为1500张（相对[5] [5] 少很多，ssd效果很差）。经过数据增强处理后得到6000张航拍图像。

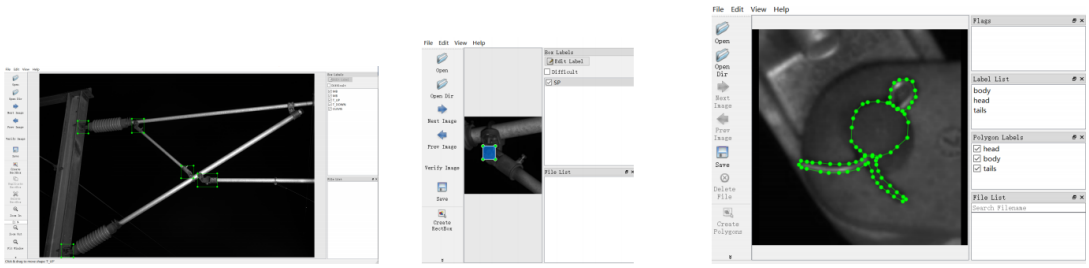
[5] [5] 提出了一种三阶段检测系统，将语义分割算法首次应用于高速铁路缺陷检测领域。该方法采用基于改进**YOLOV3**的两阶段定位方法对销钉进行定位。**第一阶段**用于定位悬链线支撑装置上的五个关节组件（桅杆支架，顶针和U形环等，这些组件里含有销钉），**第二阶段**用于定位关节组件图像中的销钉。然后，**第三阶段**实现了 **deeplabv3+** 算法对销钉进行语义分割。最后，根据销钉的头部、身体和尾部的语

义信息对销钉进行分类。

销钉有多种形状，它们安装在不同的位置。 很难有一个标准来对所有开口销进行分类。 文中将销钉根据其在悬链线上的位置和形状不同分为TYPE_A_SP、TYPE_B_SP、TYPE_C_SP三种，如图所示，可以提高分类方法的鲁棒性。 每种类型的销钉根据其自身的缺陷分类标准分为缺失、松动和正常三种状态，如图所示。



三个阶段的标注过程



(a) (b) (c)

Fig. 8. Sample dataset annotation. (a) Annotation for first localization. (b) Annotation for secondary localization. (c) Annotation for semantic segmentation.

References

1. 王凯, et al."基于辅助数据RetinaNet算法的销钉缺陷智能识别." *广东电力 32*.09(2019):41-48. doi:. ↩

2. 王凯, et al."基于RetinaNet和类别平衡采样方法的销钉缺陷检测." *电力工程技术 38*.04(2019):80-85. doi:CNKI:SUN:JSDJ.0.2019-04-013. ↩

3. 王凯, et al."基于生成对抗网络和RetinaNet的销钉缺陷识别." *华南理工大学学报(自然科学版) 48*.02(2020):1-8. doi:CNKI:SUN:HNLG.0.2020-02-001. ↩

4. Xiao, Yewei, et al. "Detection of Pin Defects in Aerial Images Based on Cascaded Convolutional Neural Network." *IEEE Access* (2021). ↩

5. Wang, Jian, et al. "A Defect-Detection Method of Split Pins in the Catenary Fastening Devices of High-Speed Railway Based on Deep Learning." *IEEE Transactions on Instrumentation and Measurement* 69.12 (2020): 9517-9525. ↩ ↩