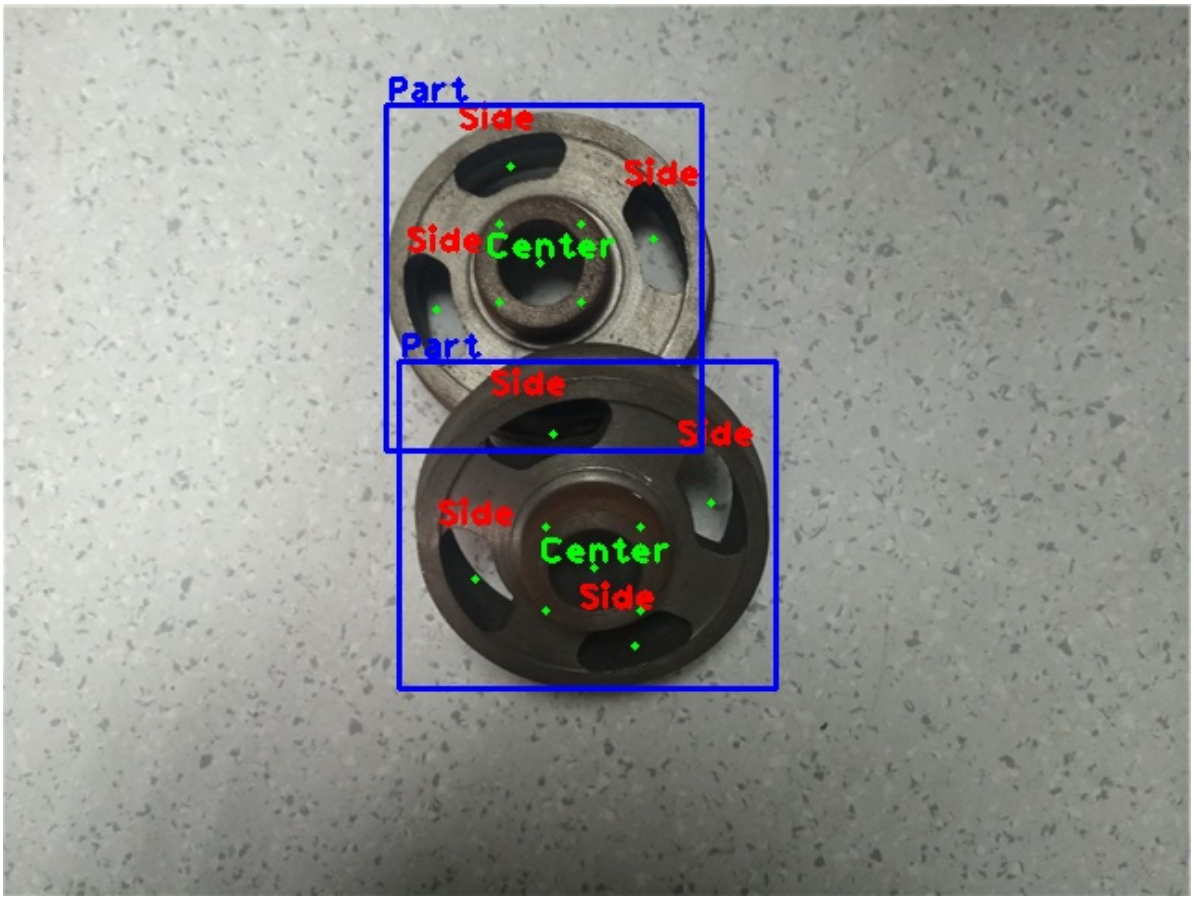
YOLO-v1网络调试、训练与测试

本报告由深港微电子学院黄冠超（学号11912309）贡献，所有相关的资源，包括源代码、数据集与报告可在我的GitHub仓库获取。

基于@abeardear提交在abeardear/pytorch-YOLO-v1 仓库中的源代码，利用YOLO-v1网络，实现对零件的主体、边缘、中心的预测与定位。



~~合并标注文件（缺少预处理）~~

> 依照原本的要求，物体及其位置标注应当从`labels`目录下的`.txt`文件中合并后读取，但随后证实其中的数据与网络训练不匹配，需要进行预处理，否则将导致模型完全失效，无法进行预测。
>
> 前期调试时，所使用的无效的标注数据可能导致诸如数值错误等未可预料的异常，在源代码调试中，部分错误可能实际上由错误的标注数据所引发。

通过文件读写合并标注数据文件。

```
path = 'labels/'
with open(path + 'label_train.txt', 'w') as fw:
    for i in range(1, 201):
        file_name = path + str(i).rjust(7, '0') + '.txt'
        with open(file_name, 'r') as fr:
            for line in fr.readlines():
                fw.write(line.strip() + ' ')
            fw.write('\n')
```

此处标注数据的组成形式为`c`, `x`, `y`, `x2`, `y2`，因此需要对应修改`dataset.py`中`yoloDataset`类下的`__init__()`方法：

```
x = float(split[1 + 5 * i])
y = float(split[2 + 5 * i])
x2 = float(split[3 + 5 * i])
y2 = float(split[4 + 5 * i])
c = split[0 + 5 * i]
```

合并标注文件

> 重构了脚本，以实现对标注的预处理，并适应网络的输入格式。

```
path = "labels/"
with open('label_train.txt', 'w') as fw:
    for i in range(1, 201):
        file_name = str(i).rjust(7, '0')
        data = file_name + '.jpg '
        with open(path + file_name + '.txt', 'r') as fr:
            for line in fr.readlines():
                obj = line.strip().split(' ')
                c = obj[0]
                obj = list(map(float, obj[1:]))
                x1 = round((obj[0] - .5 * obj[2]) * 640)
                y1 = round((obj[1] - .5 * obj[3]) * 480)
                x2 = round((obj[0] + .5 * obj[2]) * 640)
                y2 = round((obj[1] + .5 * obj[3]) * 480)
                data += '{} {} {} {} {} '.format(x1, y1, x2, y2, c)


            fw.write(data.strip() + '\n')
```

其中，190条样本用以训练，剩余10条样本用以测试。

网络源代码调试

> 如无特别说明，以下调试均在标注数据未进行预处理条件下进行，因此部分错误可能实际上由错误的数据标注所引发。

格式化代码

原作者的代码中含有大量typo、重复代码、单行代码过长、在`docstring`中使用单引号等不符合`The Zen of Python`的代码片段，且均没有进行格式化。为提升代码可读性，有必要进行相应的修复和优化。

补全`yoloDataset.encoder()`方法

依照原作者@abeardear提交在仓库 abeardear/pytorch-YOLO-v1中的源代码，修改`dataset.py`中`yoloDataset`类下的`encoder()`方法如下：

```
def encoder(self, boxes, labels):
    """
    implement the encoder
    boxes (tensor) [[x1,y1,x2,y2],[]]
    labels (tensor) [...]
    return 7x7x30
    """

    grid_num = 14
    target = torch.zeros((grid_num, grid_num, 30))
    cell_size = 1. / grid_num
    wh = boxes[:, 2:] - boxes[:, :2]
    cxcy = (boxes[:, 2:] + boxes[:, :2]) / 2

    for i in range(cxcy.size()[0]):
```

```
        cxcy_sample = cxcy[i]
        ij = (cxcy_sample / cell_size).ceil() - 1
        target[int(ij[1]), int(ij[0]), 4] = 1
        target[int(ij[1]), int(ij[0]), 9] = 1
        target[int(ij[1]), int(ij[0]), int(labels[i]) + 9] = 1

        xy = ij * cell_size
        delta_xy = (cxcy_sample - xy) / cell_size
        target[int(ij[1]), int(ij[0]), 2:4] = wh[i]
        target[int(ij[1]), int(ij[0]), :2] = delta_xy
        target[int(ij[1]), int(ij[0]), 7:9] = wh[i]
        target[int(ij[1]), int(ij[0]), 5:7] = delta_xy

    return target
```

修复图片文件无法读取的问题

> 该问题由错误的标注格式引起，后已修正。

此时运行网络训练，将会出现报错：

```
...

File "<__array_function__ internals>", line 5, in fliplr
  File "C:\ProgramData\Anaconda3\lib\site-packages\numpy\lib\twodim_base.py",
line 93, in fliplr
    raise ValueError("Input must be >= 2-d.")
ValueError: Input must be >= 2-d.
```

即，在`random_flip()`数据预处理中传入的图片数组不满足`numpy.fliplr()`的维度大于等于2的要求。

或：

```
...

File "C:\Users\Guanc\Documents\GitHub\machine-learning-lab\src\YOLO-
v1\dataset.py", line 121, in BGR2HSV
    return cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
cv2.error: OpenCV(4.0.1) C:\ci\opencv-
suite_1573470242804\work\modules\imgproc\src\color.cpp:181: error:
(-215:Assertion failed) !_src.empty() in function 'cv::cvtColor'
```

或

```
...

  File "C:/Users/Guanc/Documents/GitHub/machine-learning-lab/src/YOLO-
v1/dataset.py", line 205, in randomCrop
    height, width, c = bgr.shape
AttributeError: 'NoneType' object has no attribute 'shape'
```

等等。以上错误均出现在数据预处理环节，因各预处理执行与否为随机，因此错误也将随机出现。以上两个问题均是由图片路径为空造成，为此，在调用数据预处理方法代码前设置断点：

```
>> > if self.train:
  img, boxes = random_flip(img, boxes)
  img, boxes = randomScale(img, boxes)
  img = randomBlur(img)
  img = RandomBrightness(img)
  img = RandomHue(img)
  img = RandomSaturation(img)
  img, boxes, labels = randomShift(img, boxes, labels)
  img, boxes, labels = randomCrop(img, boxes, labels)
```

通过debugger观察到，此时，图片文件名变量`fname`值为`{str} '0'`，且`yoloDataset`对象成员变量`fnames`值为`{list: 180} ['0', ..., '0']`，读取到的图片`image`也为`None`。在对象的初始化方法中，`fnames`被初始化为

```
self.fnames = []
...
self.fnames.append(split[0])
```

可见，原作者设计的数据集读取方式与我们所指定的有所不同，图片文件名本应出现在标注数据的第一列。重新定义`fnames`列表如下：

```
idx = 1
for line in lines:
    split = line.strip().split()
    self.fnames.append(str(idx).rjust(7, '0') + '.jpg')
    idx += 1
    ...
```

即解决了图片文件无法读取的问题。

解决部分`PyTorch`语法被废弃所引起的用户警告

运行`train.py`，终端输出大量错误警告，需要一一解决。

`nn.functional.sigmoid`被废弃

```
UserWarning: nn.functional.sigmoid is deprecated. Use torch.sigmoid instead.
  warnings.warn("nn.functional.sigmoid is deprecated. Use torch.sigmoid
instead.")
```

原作者并未显式调用`nn.functional.sigmoid`，因此该警告的原因尚无从得知。

以整型进行布尔索引被废弃

```
UserWarning: indexing with dtype torch.uint8 is now deprecated, please use a
dtype torch.bool instead. (Triggered internally at
 ..\aten\src\ATen/native/IndexingUtils.h:25.)
```

该数据类型警告源于作者使用整型值`0`与`1`表征物体的存在与否，并基于此对预测值、实际值张量进行布尔型索引，而通过整型进行布尔索引将在随后的版本中被废弃。为此，只需将对应的张量转型为`torch.bool`类即可。

```
coo_mask = coo_mask.unsqueeze(-1).expand_as(target_tensor).bool()
noo_mask = noo_mask.unsqueeze(-1).expand_as(target_tensor).bool()

noo_pred_mask = noo_pred_mask.bool()

coo_not_response_mask.zero_()
coo_not_response_mask = coo_not_response_mask.bool()
```

> 值得注意的是，`PyTorch` 中并不支持类似 `NumPy` 中 `ndarray.astype()` 的张量数据类型转换，其函数返回值为转型后的张量，而原张量不变，因而必须按以下形式进行类型转换：
>
> ```
> new_tensor = tensor.bool()
> ```

由于随后在计算损失函数时将会调用上述几个张量，因此，仍然需要将其转回 `0`、`1` 整型。

`mse_loss()` 的 `size_average` 参数将被废弃

```
C:\ProgramData\Anaconda3\lib\site-packages\torch\nn\_reduction.py:44:
UserWarning: size_average and reduce args will be deprecated, please use
reduction='sum' instead.
  warnings.warn(warning.format(ret))
```

将对应的 `size_average=False` 参数改为 `reduction='sum'` 即可。

修复损失函数不可用的问题

运行 `train.py`，此时训练已能运行，但每次迭代中的损失函数均为 `nan`。

```
Epoch [1/10], Iter [5/180] Loss: nan, average_loss: nan
Epoch [1/10], Iter [10/180] Loss: nan, average_loss: nan
Epoch [1/10], Iter [15/180] Loss: nan, average_loss: nan
Epoch [1/10], Iter [20/180] Loss: nan, average_loss: nan
...
```

在 `yoloLoss.forward()` 的末尾设置断点：

```
>>> return (self.l_coord * loc_loss +
            2 * contain_loss + not_contain_loss +
            self.l_noobj * noobj_loss +
            class_loss) / N
```

通过debugger观察到，损失函数的组成部分中，`loc_loss` 为 `{Tensor} tensor(nan, grad_fn=<AddBackward0>)`，导致最终的损失函数为 `nan`。注意到 `loc_loss` 由两部分构成，将其分离进行调试。

```
loc_loss_1 = F.mse_loss(box_pred_response[:, :2],
                        box_target_response[:, :2],
                        reduction='sum')
loc_loss_2 = F.mse_loss(torch.sqrt(box_pred_response[:, 2:4]),
                        torch.sqrt(box_target_response[:, 2:4]),
                        reduction='sum')
loc_loss = loc_loss_1 + loc_loss_2
# loc_loss = F.mse_loss(box_pred_response[:, :2],
#                       box_target_response[:, :2],
#                       reduction='sum') + \
#            F.mse_loss(torch.sqrt(box_pred_response[:, 2:4]),
#                       torch.sqrt(box_target_response[:, 2:4]),
#                       reduction='sum')
```

观察到，`loc_loss_1`正常，而`loc_loss_2`值为`nan`，而用以计算`loc_loss_2`的张量`box_target_response`中出现了小于零的值，其作为`torch.sqrt()`的入参时自然会产生数值计算错误。虽然这部分负值出现的原因未知，但注意到其绝对值都极小，可以直接化为0处理。则代码对应修改为：

```
F.mse_loss(torch.sqrt(box_pred_response[:, 2:4].clamp(min=0)),
           torch.sqrt(box_target_response[:, 2:4].clamp(min=0)),
           reduction='sum')
```

至此，训练已可正常进行。

补全数据预处理方法

依照原作者提交的源代码，补全数据预处理方法如下。

随机饱和度：

```
def RandomSaturation(bgr):
    if random.random() < 0.5:
        hsv = BGR2HSV(bgr)
        h, s, v = cv.split(hsv)
        adjust = random.choice([0.5, 1.5])
        s *= adjust
        s = np.clip(s, 0, 255).astype(hsv.dtype)
        hsv = cv.merge((h, s, v))
        bgr = HSV2BGR(hsv)
    return bgr
```

随机模糊：

```
def randomBlur(bgr):
    return cv.blur(bgr, (5,) * 2) if random.random() < 0.5 else bgr
```

随机形变：

```
def randomScale(bgr, boxes):
    # fix height, scale width from 0.8 to 1.2
    if random.random() < 0.5:
        scale = random.uniform(0.8, 1.2)
        height, width, c = bgr.shape
        bgr = cv.resize(bgr, (int(width * scale), height))
        scale_tensor = torch.FloatTensor([[scale, 1] * 2]).expand_as(boxes)
        boxes *= scale_tensor
    return bgr, boxes
```

除补全方法之外，也将所有的数据预处理方法提取为`yoloDataset`之外的静态函数。

解决引入数据预处理所引发的问题

引入数据预处理方法后，在训练中产生了新的问题，Python抛出大量异常，进度条也不能正常显示。

```
Traceback (most recent call last):
  File "C:\ProgramData\Anaconda3\lib\multiprocessing\queues.py", line 239, in
_feed
    obj = _ForkingPickler.dumps(obj)
  File "C:\ProgramData\Anaconda3\lib\multiprocessing\reduction.py", line 51, in
dumps
    cls(buf, protocol).dump(obj)
_pickle.PicklingError: Can't pickle <class
'numpy.core._exceptions.UFuncTypeError'>: it's not the same object as
numpy.core._exceptions.UFuncTypeError
```

由于未提示异常具体由哪一行源代码抛出，因此只能逐行检查。通过设置断点发现，影响训练进程的代码为随机饱和度预处理。而其他的预处理方法都能正常执行。

```
img = RandomSaturation(img)
```

网络训练与测试

训练平台

- 硬件
  - 中央处理器：AMD Ryzen 7 3800X，8核16线程，标称3.89GHz，运行在4.20GHz
  - 图形处理器（训练）：ASUS TUF Gaming GeForce® GTX 1650 超频版
    - 4GB GDDR6显存，带宽12Gbps，位宽128位
    - 加速频率1680MHz
    - 896枚CUDA核心
    - 采用图灵架构
    - 连接到PCIe 4.0
  - 图形处理器（推理）：NVIDIA GeForce GTX 950，2G独立显存
  - 内存：光威深渊DDR4，标称3000MHz，运行在3000MHz，双16GB构成双通道，总32GB
  - 主板：华硕TUF Gaming Plus X570 WiFi
  - 驱动器：三星SSD 980 Pro，连接到PCIe 4.0
- 软件

- Windows 10专业版20H2，已安装所有更新
- PyCharm Professional 2020.3
- Anaconda 3中的Python 3.8.8
- PyTorch 1.8.0
- CUDA 11.2
- 其余相关包均已更新至最新版本

训练配置

> 由于GeForce GTX 1650同时作为训练平台的显示输出，需要驱动一台4K与一台2K显示器，其在空载情况下占用显存已达`0.8GB`左右，本就不大的显存愈发吃紧。经过反复测试，为避免出现显存溢出，同时尽可能优化训练效果，设定以下初始训练参数：

- 在初始化`DataLoader`时加入`pin_memory=True`参数
- batch大小设置为`2`
- 学习率初始化为`0.001`
- 在第`30`个epoch后，学习率下降至`0.0001`
- 在第`40`个epoch后，学习率下降至`0.00001`
- 共训练`50`个epoch。

在训练过程中，偶发性出现在运行多个epoch后，在某处batch结束后训练不再继续，进程也不退出。经查，此类问题往往与进程互锁有关。此项目中均采用`OpenCV`包进行数据读取，其多线程并发引起锁死的概率较大。为避免对工程进行完全重构，只需将`OpenCV`的多线程关闭即可。在`yoloDataset.__init__()`中，添加如下代码：

```
cv.setNumThreads(0)
cv.ocl.setUseOpenCL(False)
```

此时训练可正常进行，效率并未受到大的影响。

> 就机器学习而言，相较于`OpenCV`，`PIL`（`Pillow`）包往往更受推荐，因其可避免进程互锁问题，并且在`DataLoader`的数据预处理中更具灵活性。

测试配置

> 出于充分利用计算资源的考量，将网络推理放置在GeForce GTX 950上运行。

为了正常进行预测，需要对预测程序入口`predict.py`文件进行必要修改。

`torch.autograd.Variable()`的`volatile`参数被移除

在更正了图片路径后，程序运行报错：

```
C:/Users/Guanc/Documents/GitHub/machine-learning-lab/src/YOLO-v1/predict.py:140:
UserWarning: volatile was removed and now has no effect. Use `with
torch.no_grad():` instead.
  img = Variable(img[None, :, :, :], volatile=True)
```

根据提示修改代码如下：

```
with torch.no_grad():
    img = Variable(img[None, :, :, :])
    img = img.cuda()
```

重构类名标签集

为了正确输出分类标签，需要重写类名标签集。

```
CLASSES = ('part', 'center', 'side')
...
result.append([(x1, y1), (x2, y2), CLASSES[cls_index], image_name, prob])
```

解决不能正确输出预测结果的问题

> 该问题后证实为数据标注错误所引起。

执行脚本进行推理，发现输出的图片中并未给出任何分类、定位方框与文字。此时，程序输出的`result = predict_gpu(model, image_name, root_path=root_path)`的值始终为`{list: 1} [[(0, 0), (0, 0), 'part', '<filename>', 0.0]]`，换言之，没有侦测到物体以及定位框。

解决非零重载被废弃的用户警告

运行预测，Python抛出用户警告：

```
C:/Users/Guanc/Documents/GitHub/pytorch-YOLO-v1/predict.py:165: UserWarning:
This overload of nonzero is deprecated:
 nonzero()
Consider using one of the following signatures instead:
 nonzero(*, bool as_tuple) (Triggered internally at
 ..\torch\csrc\utils\python_arg_parser.cpp:882.)
 idx = (iou <= threshold).nonzero().squeeze()
```

依照提示修改参数：

```
idx = (iou <= threshold).nonzero(as_tuple=False).squeeze()
```

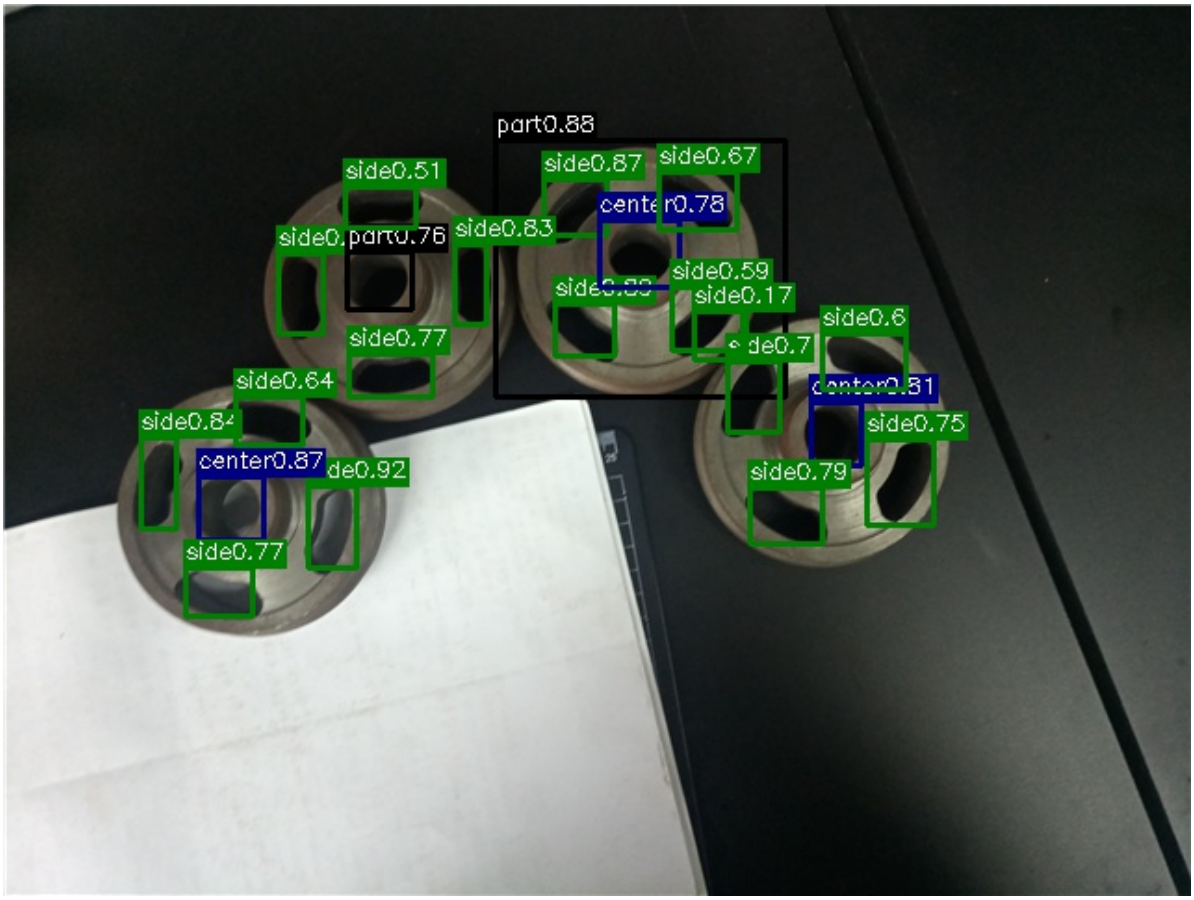
无预处理条件下训练

第一次训练（无预处理、50代）

以初始训练设置进行训练，在不进行数据预处理时，针对190个样本进行学习，耗时1308030毫秒，约合21.80分钟，在训练集上得到的平均损失为`9.90`。

> 在关闭风扇时，GPU的满负荷运行温度达到80℃；而风扇全速运转时，下降至约50℃，可见风扇启停对GPU性能与训练效率有重要影响。

依照网络测试中的设置进行预测，下图为对训练集的最后一个样本`0000190.jpg`进行预测的结果，测试集上的效果也类似。可见，虽然整体效果较好，但其常将零件的中心识别为零件主题，仍有欠拟合现象。



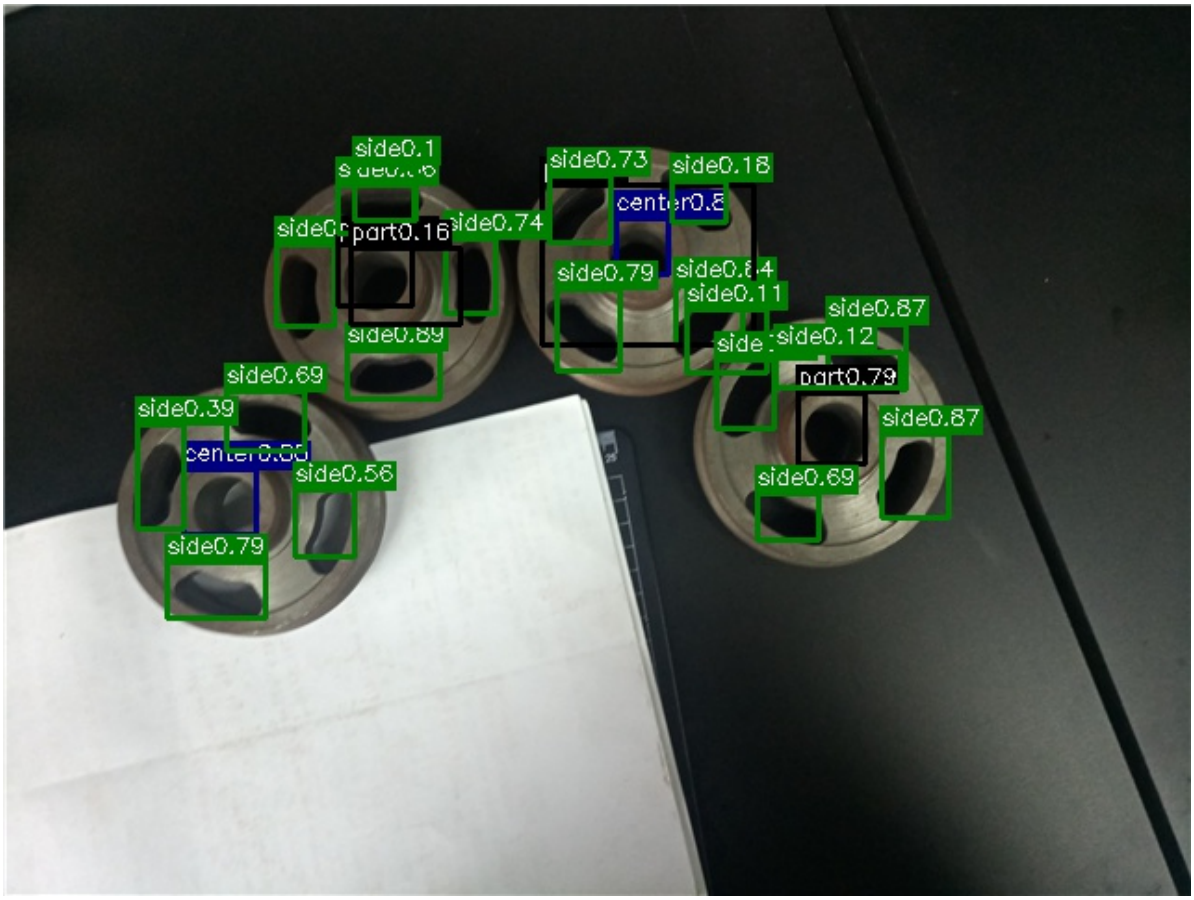
第二次训练（无预处理、50代、提高学习率）

针对欠拟合现象，适当提高学习率，作如下调整：

- 学习率初始化为 `0.005`
- 在第 `30` 个epoch后，学习率下降至 `0.001`
- 在第 `40` 个epoch后，学习率下降至 `0.0001`

经训练，最后在测试集上获得的平均损失下降至 `8.45`.

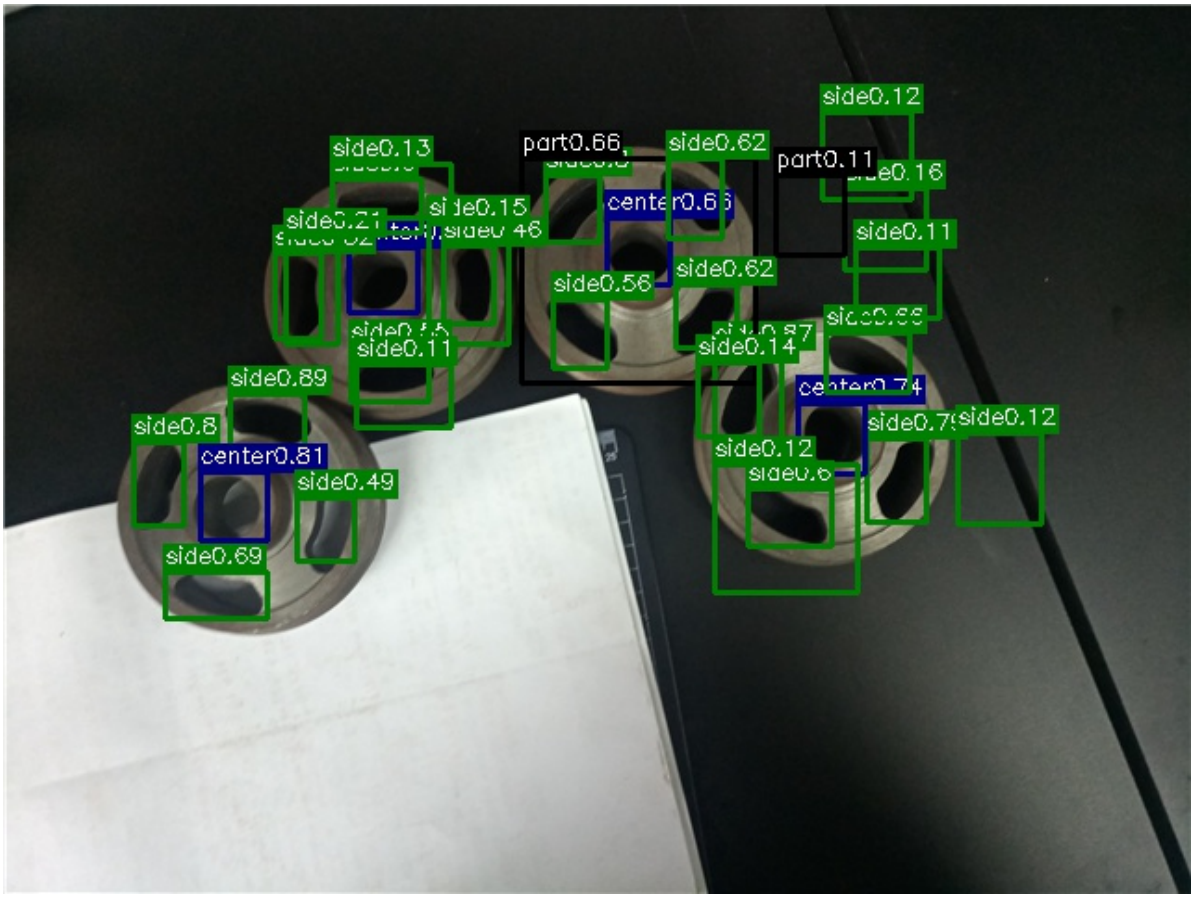
下图为对训练集的最后一个样本 `0000190.jpg` 进行预测的结果。可见，识别效果仍然不尽如人意，甚至相较于第一次训练出现了更多的重复识别边缘的现象。在测试集上，效果也类似。



第三次训练（无预处理、100代）

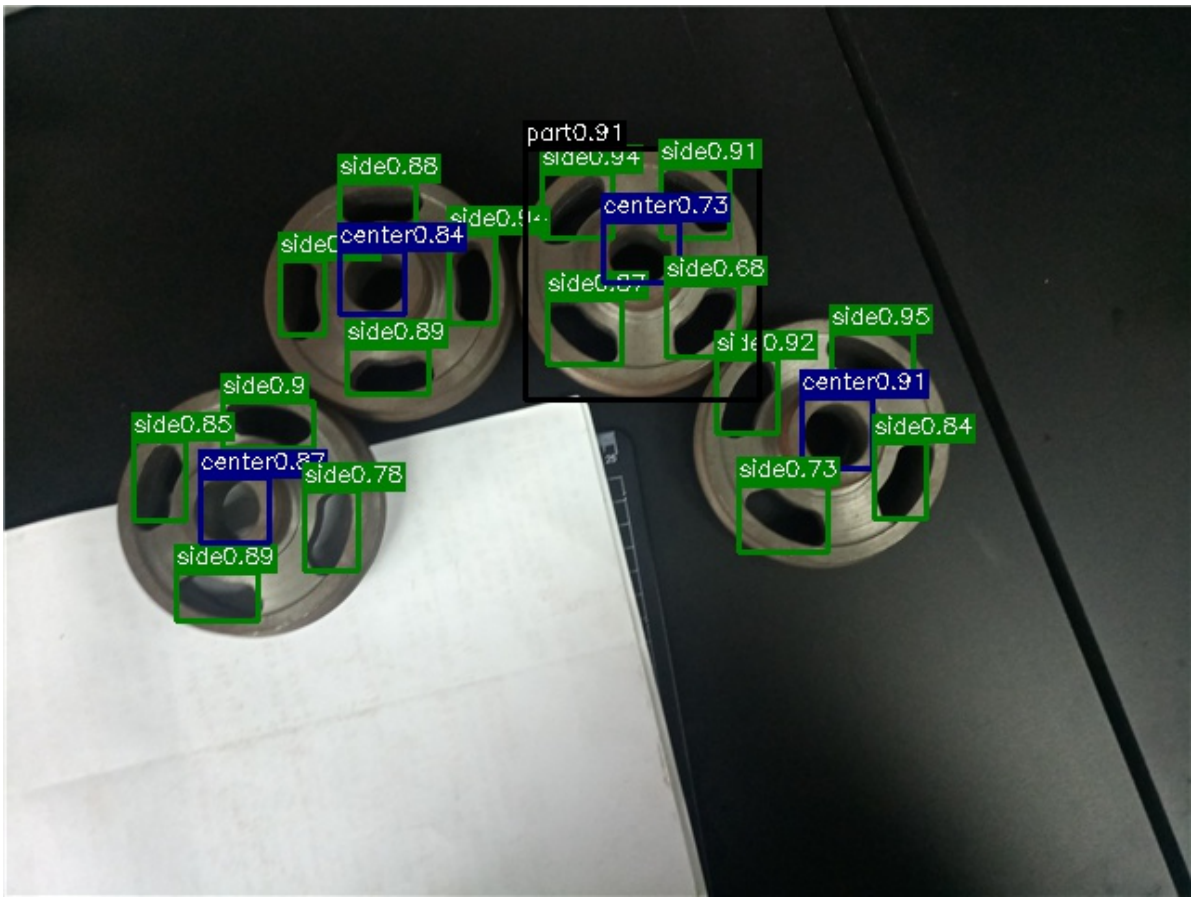
进一步增加epoch至100代，降低学习率为`0.001`在测试集上获得的损失反而上升至`9.96`，观察预测结果。

下图为对训练集的最后一个样本`0000190.jpg`进行预测的结果。

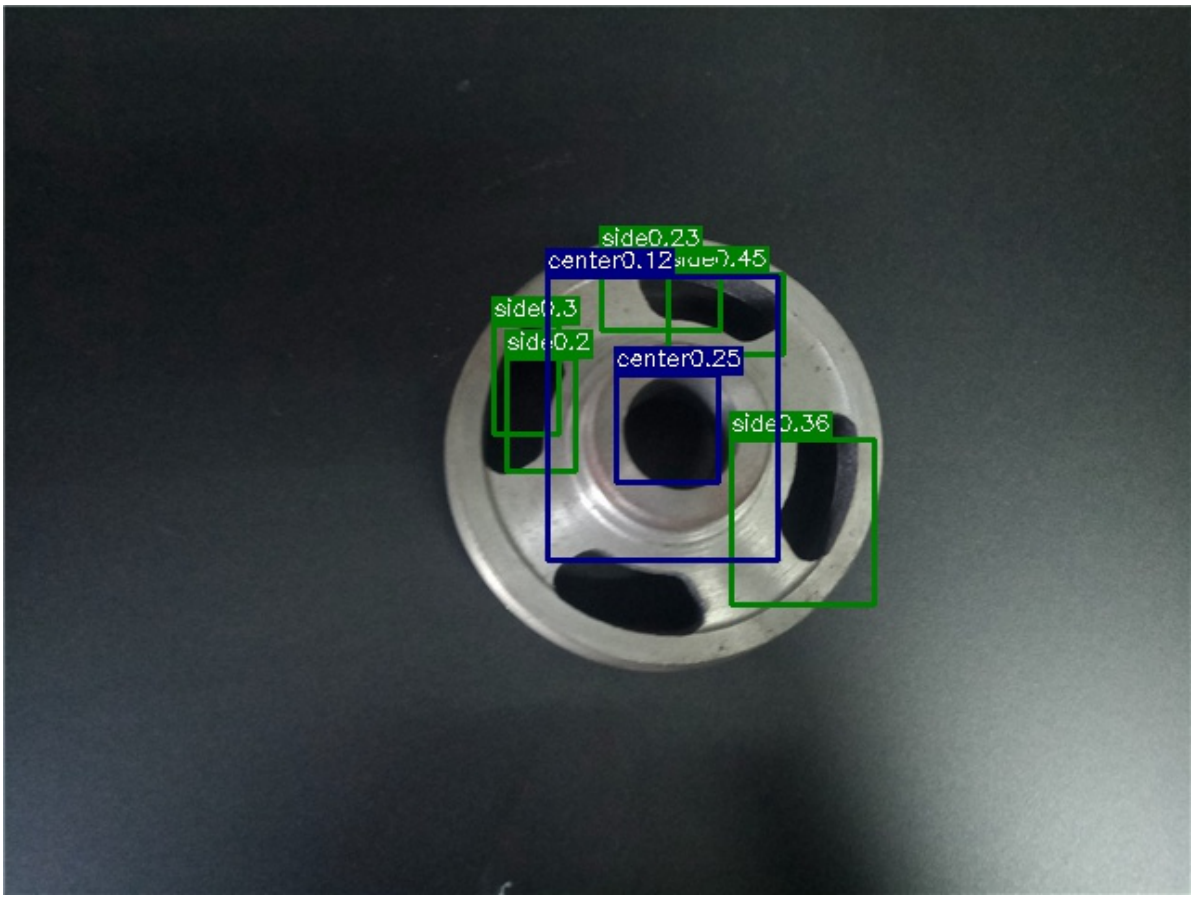


可见，出现了大量错误的物体识别。将检测阈值从`0.1`提高至`0.2`，得到结果如下：

```
# mask1 = contain > 0.1
mask1 = contain > 0.2
```



但是，在测试集上，仍有错误产生。



> 至此，需要考虑通过在训练时认为加入噪声，以提升网络的预测能力。

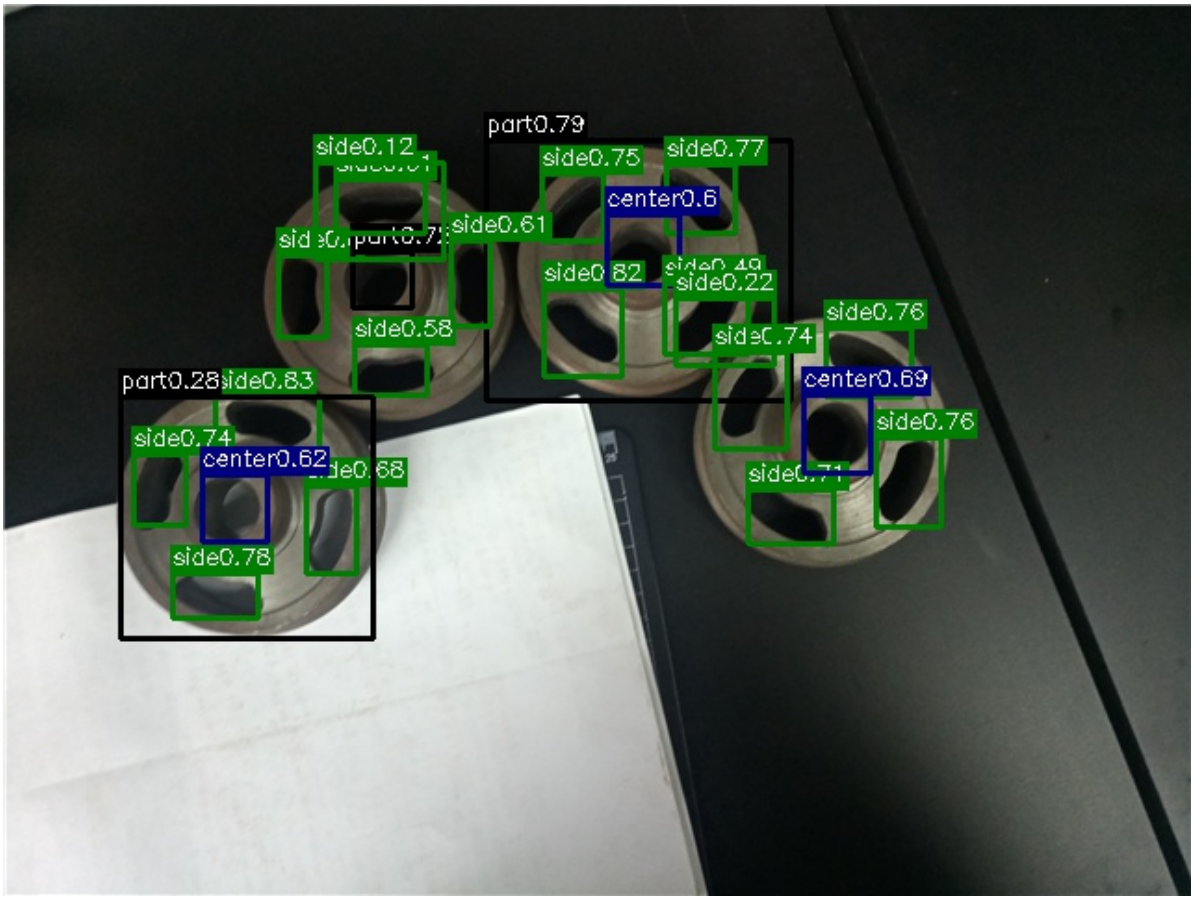
结合数据预处理进行网络训练

第四次训练（预处理，80代）

在引入了图片预处理后，对模型进行训练。

- 学习率初始化为`0.005`
- 在第`40`个epoch后，学习率下降至`0.001`

总训练用时2144026毫秒，约合35.73分钟，在测试集上得到平均损失为`2.78`。



总的来讲，虽然此时效果已较之前三次训练有大的提升，但仍然存在边界识别重复、混淆零件主体和零件中心的问题。

总结

通过对源代码的大量调试，以及对训练、推理参数的试错优化，成功实现了网络的训练、预测全流程，效果较好，但仍不能达到为完美，有待进一步的优化。