

哈尔滨工业大学

<<模式识别与深度学习>>

大作业 目标检测算法探究

(2020 春季学期)

学院： 计算机科学与技术

学号： 1170300909

1170300906

姓名： 武磊

吴月明

指导老师： 左旺孟

日期： 2020.4.26

一、实验目的

探究目前常用的基于深度学习的目标检测方案，理解其基本方法论和想法，在此基础上，针对具体的目标检测问题，实现模型。

二、实验环境配置

1. 硬件：

CPU : Inter Core i7-7500U 2.70GHz

GPU : NVIDIA GeForce 940MX

2. 软件：

操作系统:WIN10

软件：

Python 3.6 (基于 Anaconda)

Pytorch 1.2.0

CUDA 10.1 V10.1.105

cudnn v7.6.3 for CUDA 10.1

3. 开发 IDE

Jupyter NoteBook

Pycharm

三、背景调研

(一) 问题提出

我们想从 kaggle 数据竞赛中找到一些想法，原因有以下两点：

- i) 解决了数据集的问题，kaggle 竞赛的主办方往往会提供完备的数据集，用于测试和训练，这对于深度学习是重要的前提。
- ii) 这里有丰富的社区资源，能够提供好的解决方案。同时有科学的测试方法，能够很好的衡量模型的效果

我们选择的是 kaggle 上的 Global Wheat Detection 任务，小麦头检测任务。

(二) 方法调研

目标检测是很经典的高层视觉任务，在深度学习方法提出之前就有一系列的传统方法，我们主要考虑深度学习的检测方法。

基于深度学习的目标检测主要分成两大类，one stage 目标检测和 two stage 阶段目标检测过程。

Two stage 目标检测的思想很简单：

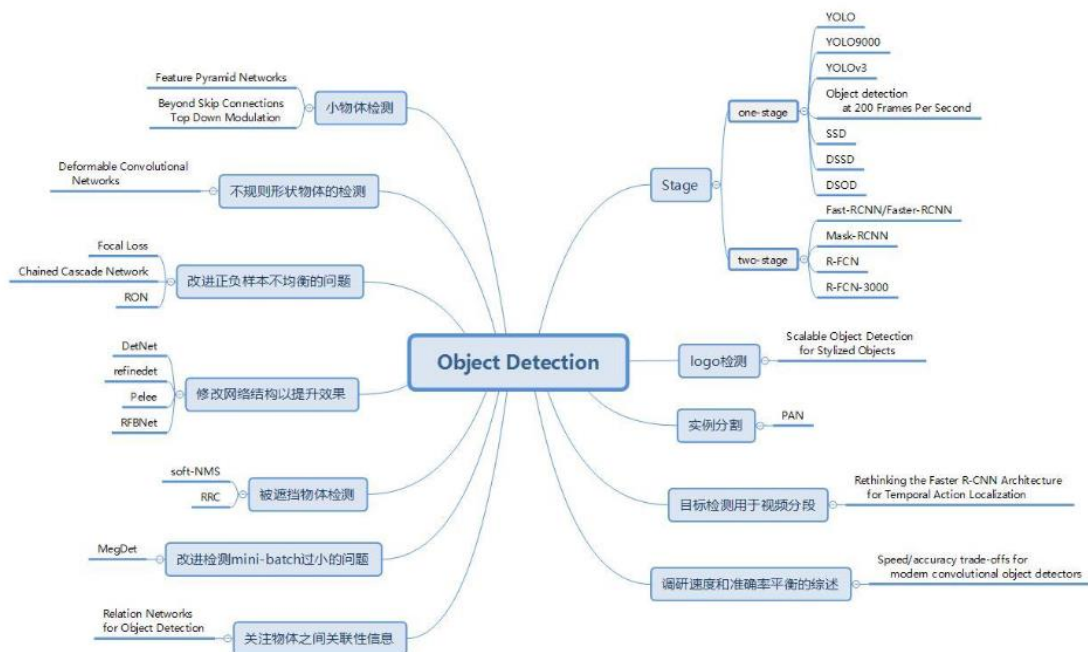
i) First stage: Region Proposal

ii) Second Stage: Classification

常见的 two stage 检测方法主要以 RCNN 系列为主，一般基于锚点遍历所有可能的候选框，通过神经网络或者启发式方法从候选框中筛选 ROI (Region of Interest)，送入分类网络中对每一个框中的物体进行分类。

常见的 one stage 方法主要 Yolo (You only look once) 系列和 SSD (Single shot detector), 其检测方案是均匀地在图片的不同位置进行密集抽样, 抽样时可以采用不同尺度和长宽比, 然后利用 CNN 提取特征后直接进行分类与回归, 整个过程只需要一步。

下面是一张目标检测最新的进展图:



我们主要调研和比较了各种方法的优劣和实现训练的难度, 同时考虑到具体任务的要求, 做了以下尝试:

- i) 调用 `torchvision.model.detection` 中的实现, 在训练集上训练了 `resnet50-fpn-fasterrcnn` 网络, 同时测试效果, 获得了较为理想的检测效果, 主要问题是该网络的实时检测速度太慢。
- ii) 尝试实现 `anchor free` 的单阶段检测算法 `centernet`, 该网络的思想简单, 但是效果强大, 实现和训练难度也适中。
- iii) 在测试 `centernet` 效果之后尝试修改其 `backbone` 结构, 试图获得更好的测试效果。

(三) 算法原理

1. Faster RCNN+FPN 原理

1.1 RCNN

不同与分类任务, 目标检测需要在图片中框选出目标的位置。一种简单的方式是使用滑动窗口依次扫描整张图片进行目标定位, R-CNN 采用的是一种对滑动窗口改进的思想。类似于滑动窗口, R-CNN 选取出 2000 个可能的窗口区域(Region Proposals), 对其进行特征提出, 然后通过 SVM 对其进行分类。

滑动窗口的思想基本上穷举了图像中物体所有出现的区域框, 复杂度太高, 同时目标的尺寸多样, 对于窗口的大小不好控制。论文中采用 `Selective search` 的方式进行 `Region Proposals` 减少了候选框的数量。`Selective Search` 的基本思想是: 先分割出小尺度的区域, 然后进行合

并。

通过上述的 Region Proposals 得到了若干个候选框，我们将它们均输入到一个 CNN 中最后得到一个 4096 维的特征向量中。由于候选框大小不一，而 CNN 的输入大小为 227x227，故需要将所有候选框转换到一个相同的尺寸。有三种拉伸方式：1. 裁剪。2. 按比例拉伸，空闲部分用零填充。3. 直接拉伸（图片比例可能变化）。

接下来通过 SVM 对特征进行分类打分，对所有打分的候选框，采用非极大抑制选出候选框。使用 SVM 而不使用 softmax 层的原因是 softmax 的效果与 SVM 相差 4%，fine-tuning 过程中使用了大量 IOU 在 0.5-1 之间的数据，这些数据的使用是为了防止 CNN 过拟合，但是这些数据没有特别注重准确的定位信息，softmax 则对这些不准确的框进行了训练，但是 SVM 的标准更加严格。当然 SVM 会加大时间消耗，这是时间和准确度上的平衡。在 SVM 进行分数计算后，采用了一个线性回归算法提升框选位置的准确性。

CNN 采用的是对 ILSVRC2012 分类数据集的预训练模型。CNN 的参数针对拉伸折叠的候选区域进行了 fine-tuning。fine-tuning 选取与 ground-truth 的 IOU 大于 0.5 的为正例，否则则标为背景。SVM 训练过程中只选取 ground-truth 为正例，与 ground-truth 的 IOU 小于 0.3 的为负例（背景），其他的忽略。

缺点：1. Region Proposals 算法是固定的，不能针对图片动态调整。2. 非端到端。3. 训练过程极慢，需要磁盘空间大。4. 检测阶段慢

1.2 Fast RCNN

Fast RCNN 的框架是对整张图输入 CNN 和一个 max pooling，对 feature map 使用 region proposals，对每个 RoI 通过一个 RoI pooling 层得到若干个固定大小的特征向量，将特征向量输入全连接层，然后输入到 2 个输出层，一个输出层输出 softmax 概率信息，一个输出层对 bounding-box 进行调整。

RoI Pooling Layer 对不同尺度的候选框框输出一个固定大小的特征向量。

1.3 Faster RCNN

Faster RCNN 的 CNN 部分采用 ZFnet 或 VGG，Faster RCNN 采用了 Region Proposal Networks 进行候选框的预测。RPN 输入一张任意大小的图像，输出一系列矩形候选框，并为每个候选框分配一个分数（衡量 object 和 background 的囊括程度）。RPN 用一个小的网络以滑动窗口的形式扫描整个由第一个 CNN 输出的特征图，将其映射为一个相对低维的特征（ZF: 256, VGG: 512）。然后将该特征输入到两个全连接层中 box-regression layer 和 box-classification layer。

我们对输入 RPN 的 feature map 上的每一个点设为一个锚点，设每个锚点对应 k 个候选框，则对于 box-regression layer 会输出 4k 大小的输出，分别代表这 k 个候选框的需要修正的坐标差，box-classification layer 会输出 2k 大小输出，为是否是目标的概率。锚点虽然是由特征图上定义的，但它实际上应该是原始图像上的点（比如 VGG 下采样比为 16，那么原图像每两个 anchor 相隔 15 个像素）。faster RCNN 中由 9 个 anchor boxes，分别是三种不同大小(128,256,512)

和三种不同长宽比(1:1,1:2,2:1)，来模拟不同尺度和比例的 bbox。

$$\text{LOSS 函数 } L(p_i, t_i) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*)$$

1.4 FPN

图像的特征金字塔利用了每层的信息，但是对于运算时间成本也是加倍的。深层的卷积神经网络就是一个金字塔结构，每一层传递地计算特征信息。卷积神经网络可以通过不同的深度输出不同分辨率的图像，但是输出结果和输入结果的语义信息差异逐渐变大。底层的是高分辨率特征图，含有的语义信息（抽象程度）少，高层含有的语义信息多。

- 1.自底向上，前向计算。将输出同样大小的卷积层视为同一个 stage，选取每个 stage 中最后一层进行金字塔的构建，因为最后一层拥有的特征信息更多。对于 ResNet 是将每个残差模块的最后一层提取出来。
- 2.自上而下，自上而下即将小分辨率特征图的语义信息和特征与大分辨率特征图相结合。结合的形式是，将大分辨率的特征图通过 1x1 的卷积使得它与小分辨率的特征图具有相同的通道数，将小分辨率的特征图进行 2 倍上采样，然后两者相加。

2. CenterNet 原理

2.1 简介

目标识别一般分为 one-stage 和 two-stage 两类，在此部分实验中采用的 One-stage 的 CenterNet。传统的目标检测方法主要通过锚点得到 bounding-box，而 CenterNet 抛弃了这种思想，通过预测中心点的位置以及框的长宽，然后得到 bounding-box。CenterNet 相比传统的方法具有无需后处理，速度快，在使用合适的主干骨网络后可以达到 two-stage 网络的效果。

2.2 总体思想

2.2.1 Heatmap

将为每个物体寻找合适的框的问题，转换为寻找物体中心的问题。我们将整张图片视为以物体中心而发散的高斯热力图。

2.2.2 Heatmap 预测

我们以 ground-truth 的 bounding-box 的中心为每张图建立热力图，然后输入到预测网络中进行训练。对于预测网络中热力图生成的 loss 函数为：

$$L_k = -\frac{1}{N} \sum_{xyc} \begin{cases} (1 - \hat{Y}_{xyc})^\alpha \log(\hat{Y}_{xyc}) & \text{if } Y_{xyc} = 1 \\ ((1 - Y_{xyc})^\beta (\hat{Y}_{xyc})^\alpha \log(1 - \hat{Y}_{xyc})) & \text{otherwise} \end{cases}$$

Loss 是以 focal loss 的形式，其中 \hat{Y}_{xyc} 为网络预测的热力图， Y_{xyc}

为 ground-truth 的热力图，focal loss 是交叉熵函数的一种改进方式，通过添加指数给予不同输出不同的权值，对于正例：预测靠近 1 的权值较低，而预测效果差的权值较高，对于反例同理。Focal loss 提升了对于难以区分的目标的权值，降低了以及能够很好预测目标的权值，增强了网络的训练能力。

2.2.3 中心点偏移预测

由于保持原图分辨率大小的计算复杂度过高，故会对模型进行一

定程度的下采样操作，下采样操作过程对于中心点会通过除以采样比得到下采样后图像的中心点。通常情况下，我们为了能够在图像中标识出中心点会对除后的数进行取整操作，因此带来了不可避免的误差。CenterNet 中引入了偏移量，对该误差进行修正。在网络中，我们同时也需要预测一个 x 和 y 作为中心点的偏移量，它的 loss 函数为：

$$L_{off} = \frac{1}{N} \sum_p |\hat{O}_{\tilde{p}} - (\frac{p}{R} - \tilde{p})|$$

其中 $\tilde{p} = \lfloor \frac{p}{R} \rfloor$ ，即 $\frac{p}{R}$ 为一个浮点数，而 \tilde{p} 为一个整数，以 L1 loss 的

形式对其进行了约束。 $\hat{O}_{\tilde{p}}$ 为网络输出的偏移量的值，我们希望它能够靠近该由于下采样所带来的误差。

2.2.4 B-box 大小预测

在得到中心后，我们需要框的 w 和 h 值得到确切的框，同样使用相同的预测网络得到 w, h 值，然后以大小的形式进行 loss 约束：

$$L_{size} = \frac{1}{N} \sum_{k=1}^N |\hat{s}_{pk} - s_k|$$

其中 s_k 为 ground-truth 的大小， \hat{s}_{pk} 为预测结果的大小，网络实际

输出为 \hat{w}_{pk} 和 \hat{h}_{pk} ，通过相乘得到预测的面积。

2.2.5 网络的 loss



$$L_{det} = L_k + \lambda_{size} L_{size} + \lambda_{off} L_{off}$$

网络的 loss 为上述三个 loss 的加权和的形式

四、实验过程

(一) 预处理

实验数据是图片加 csv 文件索引的形式组成，首先对文件进行解析

| ▲ image_id | # width | # height | ▲ bbox | ▲ source |
|-----------------------|---|---|-----------------------------|--|
| 3373 unique values |  |  | 147761 unique values | ethz_1 35% arvalis_1 31% Other (50588) 34% |
| b6ab77fd7 | 1024 | 1024 | [834.0, 222.0, 56.0, 36.0] | usask_1 |
| b6ab77fd7 | 1024 | 1024 | [226.0, 548.0, 130.0, 58.0] | usask_1 |
| b6ab77fd7 | 1024 | 1024 | [377.0, 504.0, 74.0, 160.0] | usask_1 |
| b6ab77fd7 | 1024 | 1024 | [834.0, 95.0, 109.0, 107.0] | usask_1 |
| b6ab77fd7 | 1024 | 1024 | [26.0, 144.0, 124.0, 117.0] | usask_1 |

通过 process_bbox 函数对 bbox 进行解析

```
1. def process_bbox(df):
2.     df['bbox'] = df['bbox'].apply(lambda x: eval(x))
```

```

3.     df['x'] = df['bbox'].apply(lambda x: x[0])
4.     df['y'] = df['bbox'].apply(lambda x: x[1])
5.     df['w'] = df['bbox'].apply(lambda x: x[2])
6.     df['h'] = df['bbox'].apply(lambda x: x[3])
7.     df['x'] = df['x'].astype(np.float)
8.     df['y'] = df['y'].astype(np.float)
9.     df['w'] = df['w'].astype(np.float)
10.    df['h'] = df['h'].astype(np.float)
11.
12.    df.drop(columns=['bbox'],inplace=True)
13. #     df.reset_index(drop=True)
14.    return df

```

然后对图片进行图像增广

```

1. def get_transforms(phase):
2.     list_transforms = []
3.     if phase == 'train':
4.         list_transforms.extend([
5.             Flip(p=0.5)
6.         ])
7.         list_transforms.extend(
8.             [
9.                 ToTensor(),
10.            ])
11.         list_trfms = Compose(list_transforms,
12.                               bbox_params={'format': 'pascal_voc', '
label_fields': ['labels']})

```

将 COCO 形式的数据集转换为 Pascal_voc 的形式

(二) Faster RCNN 数据集制作

继承 torch 的 Dataset 类，重写 __len__ 函数和 __getitem__ 函数，根据 data frame 中的图像 id 进行图片的索引，从文件夹中进行读取。计算图片中的 bbox，并通过 albumentations 库进行图片的增广操作，最后返回图片的 tensor 形式，bbox 和图片对应的 id

(三) CenterNet 数据集制作

3.1 draw_umich_gaussian 函数

输入图片对应的矩阵和中心，以中心生成一个指定半径的高斯热力图，通过高斯函数（gaussian2D 函数）计算图中心周围热力值

```

1. def gaussian2D(shape, sigma=1):
2.     m, n = [(ss - 1.) / 2. for ss in shape]
3.     y, x = np.ogrid[-m:m + 1, -n:n + 1]
4.
5.     h = np.exp(-(x * x + y * y) / (2 * sigma * sigma))

```

```

6.     h[h < np.finfo(h.dtype).eps * h.max()] = 0
7.     return h

```

通过对边缘和指定半径的裁剪，将中心设为 1，其他部分调用高斯函数进行求值，在多个高斯函数在此处均有热力值时，去其中的最大值。

`np.maximum(masked_heatmap, masked_gaussian * k, out=masked_heatmap)`

效果如下：

```

[[0.97632267 0.39361463 0.40221674 0.6611573  0.10836802]
 [0.61131069 0.16901332 0.32919299 0.41111229 0.67304878]
 [0.24101545 0.64511223 0.8378578  0.8007374  0.64118039]
 [0.89740317 0.41111229 0.8007374  1.          0.8007374 ]
 [0.10836802 0.76812966 0.69456836 0.8007374  0.64118039]]

```

3.2 Dataset

继承 `torch` 的 `Dataset` 类，重写 `__len__` 函数和 `__getitem__` 函数，根据 `data frame` 中的图像 `id` 进行图片的索引，从文件夹中进行读取。对图片进行随机翻转（仿射变换，调用 `get_affine_transform` 函数）

```

1. trans_input = get_affine_transform(c, s, 0, [input_w, input_h])
2.     inp = cv2.warpAffine(img, trans_input, (input_w, input_h), f
    lags=cv2.INTER_LINEAR)
3.     inp = (inp.astype(np.float32) / 255.)

```

然后调用 `color_aug` 函数对图像进行颜色变换，增强鲁棒性

`color_aug(self._data_rng, inp, self._eig_val, self._eig_vec)`

接下来将图像归一化

```

1. inp = (inp - self.mean) / self.std
2.     inp = inp.transpose(2, 0, 1)

```

将图片的大小修正为 512*512（原始图片大小为 1024*1024）

最好通过 `draw_umich_gaussian` 生成 heatmap 的 ground-truth

```

1. radius = gaussian_radius((math.ceil(h), math.ceil(w)))
2.     radius = max(0, int(radius))
3.     ct = np.array(
4.         [(bbox[0] + bbox[2]) / 2, (bbox[1] + bbox[3]) /
    2], dtype=np.float32)
5.     ct_int = ct.astype(np.int32)
6.     draw_umich_gaussian(hm[0], ct_int, radius)

```

返回缩放后的图像，heatmap 的 ground-truth，对应的偏移的索引，bbox 的大小，缩放形式，原始的 bbox 用于 mAP 的计算

(四) Faster RCNN 模型结构

从 `torchvision.models.detection` 中导入预训练的，以 `resnet50` 为主干

骨的 faster rcnn 以及 FPN 模块。获取输入 box-classifier layer 中的特征维数, 修改 box-classifier layer 中的分类数, 将类别改为两类(背景和目标), 将预测器进行替换

```
1. # eval and submit submission
2. model = torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained=False)
3. num_classes = 2 # 1 class (wheat) + background
4.
5. # get number of input features for the classifier
6. in_features = model.roi_heads.box_predictor.cls_score.in_features
7.
8. # replace the pre-trained head with a new one
9. model.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes)
```

(五) CenterNet 模型结构

(六) CenterNet 后处理

(七) mAP 计算

mAP 是用来衡量目标检测效果最常用的指标。在了解 mAP 的概念和计算之前, 首先需要明确几个概念。

i) 准确率和召回率

准确率是指在所有检测为阳性的样本中, 确实为阳性的概率, 即为真阳率。

召回率则是在所有的阳性的样本中, 被检测器检测出来为阳性的比例。

$$Precision = \frac{TP}{TP + FP}$$

TP = True positive

TN = True negative

$$Recall = \frac{TP}{TP + FN}$$

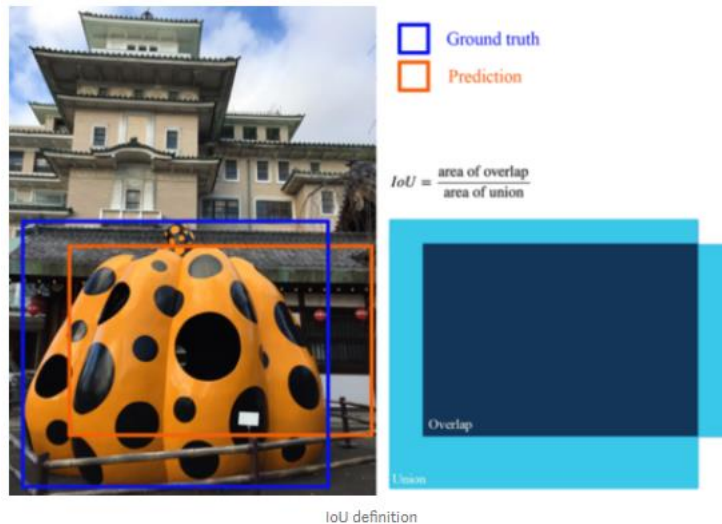
FP = False positive

FN = False negative

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

ii) IOU 交并比

IOU 则是用来衡量两个检测框之间重合的程度。显然预测框和 Ground Truth 重合程度越高, 则目标检测器的预测精度更高。



iii) AP (Average Precision)

AP 是对于每一个类别而言的，简单一些的理解方式是对于每一个类别，计算 precision-recall 曲线下方的面积。

具体实现：

- i) 首先要对模型的预测结果进行排序，按照预测值的置信度来降序排列
- ii) 对于给定的 rank, precision 和 recall 只在高于其 rank 的预测值中计算。
- iii) 计算上述得到的多个 recall, precision 的平均值

iv) mAP

对于多类别的目标检测，只需要对各个类别的 AP 求均值即可得到相应的多类别目标检测器的 mAP 值。

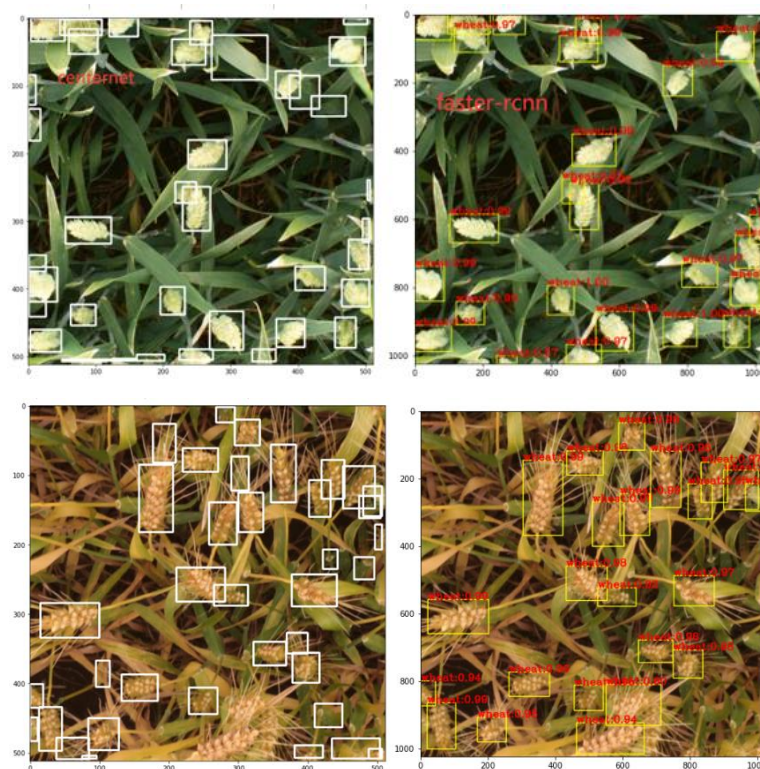
五、结果分析

在本次实验中，我们完全实现了 anchor-free 的 centernet 网络架构，包括数据预处理，端到端网络架构，编解码，训练，测试等步骤。同时，作为对比，我们采用了 anchor-based faster-rcnn 网络。并在 Global Wheat Detection 上进行了测试。

测试结果如下：

| 网络 | Backbone | mAP@0.5 | mAP@0.75 | Model size |
|-------------|--------------|---------|----------|------------|
| Centernet | Resnet50 | 0.552 | 0.249 | 349M |
| Centernet | Resnetro-fpn | 0.57 | 0.308 | 879M |
| Centernet | Hourglass | 0.576 | 0.32 | 1095M |
| Faster-RCNN | Resnet50-rpn | 0.748 | 0.42 | 314.71M |

预测效果图：



分析：

1. 对于 **centernet**，基本的网络是依据 **resnet50**，先进行五次下采样，原图像从 **512*512** 下采样到 **16*16**，这一步是为了提取图像的语义特征。之后通过转置卷积（逆卷积）操作，将原图像上采样到 **128*128** 大小，这一步是恢复图像的空域特征，保证在进行 **heatmap** 预测时候，尽量能够更加精准。
基于此观点，我们改进了 **backbone** 网络。
第一次改进，是不单纯的进行逐层上采样，而是在上采样过程中，进行跨层连接，有利于保留语义特征。
第二次则是参考了 **cornernet** 中使用的 **hourglassnet**，是更为复杂的网络，进行了多次上采样下采样的过程，作者相信这些过程有利于更好的结合空域特征和语义特征。
2. 经过实验验证：在 **GWD** 数据集上，我们发现，对于前后背景差较大的图像，我们的 **centernet** 模型能够获得较为优良的检测效果。但是对于前后背景差较小的图像，**heatmap** 的预测效果较差，很容易出现 **false-positive** 的区域。而 **fasterrcnn** 能够获得更好的实验效果。
3. 我们同时测试了两种网络的耗时：

Faster-Rcnn:

```
# inference
tic = time.perf_counter()
for images, image_ids in test_data_loader:
    dur = time.perf_counter() - tic
    images = list(image.to(device) for image in images)
    outputs = model(images)
    print('using: {}ms'.format(1000*dur/len(image_ids)))
    tic = time.perf_counter()

using: 1770.250590001524ms
28
using: 83.2087680028053ms
25
using: 162.36321500036865ms
25
```

Centernet:

```
t0 = time.perf_counter()
for sample in test_loader:
    for k in sample:
        if k != 'img_id':
            sample[k] = sample[k].to(device)

    with torch.no_grad():
        img = sample['image']
        img_id = sample['img_id'][0]

        output = model(img)[-1]
        print('using: {}ms'.format(1000*dur))
```

```
using: 92.032024749642ms
using: 38.357305500085204ms
using: 71.94136050020461ms
```

六、参考

【1】 mAP Intoduction <https://www.jianshu.com/p/ba1f7895b429>