手把手教你完成Faster-RCNN训练

该项目是fork了B站大佬bubbliiiing的，为了学习双阶段目标检测，有一部分注释也是自己重写了。但尊重原著，有需要的可以看原代码链接。

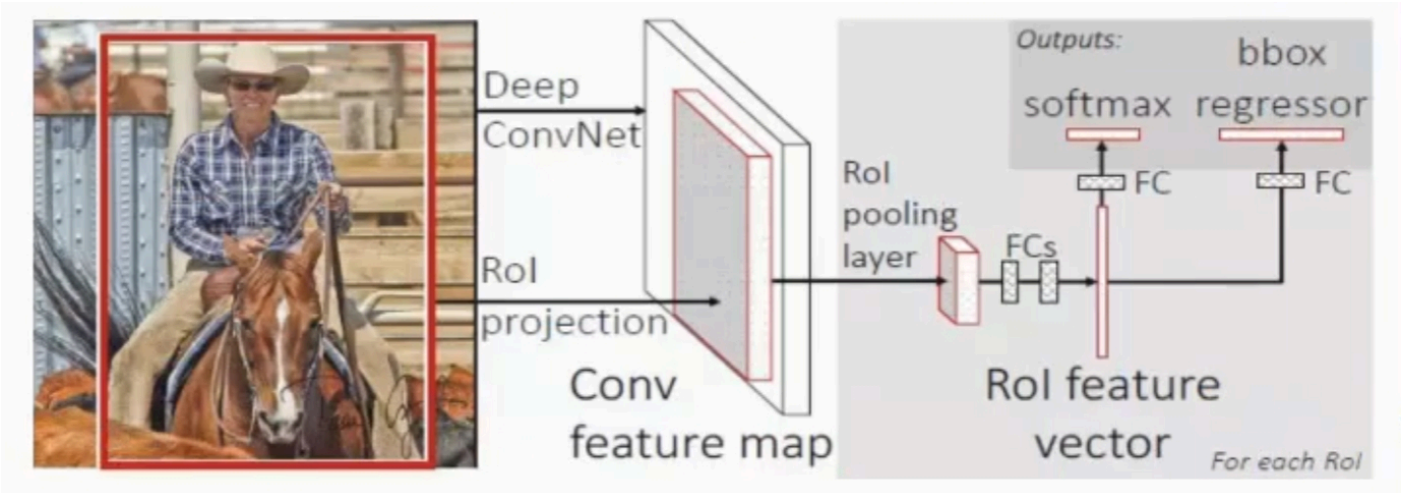
1. 文件目录介绍

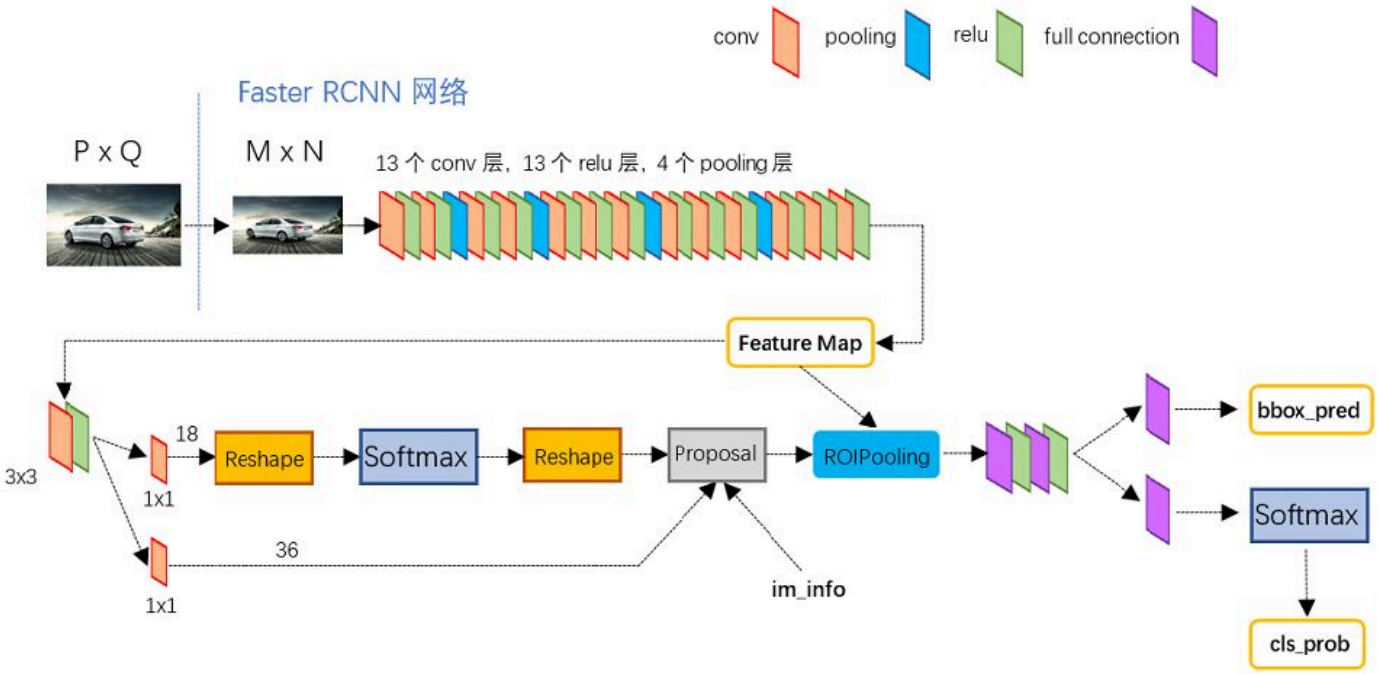
```
-- \faset-rcnn
  -- \networks
      -- \model_data
      -- resnet50.py
      -- rpn.py
      -- frcnn.py
      -- frcnn_training.py
      -- classifier.py
  -- \utils
      -- anchors.py
      -- utils_bbox.py
      -- dataloader.py
      -- utills.py
      -- utils_map.py
      -- utils_fit.py
      -- callbacks.py
  -- \dataset
      -- VOC2007
        -- Annotations
        -- lmageSets
        -- JPEGlmages
        -- SegmentationClass
        -- SegmentationObject
  -- train.py
  -- voc_annotation.py
  -- get_map.py
  -- predicet.py
  -- summary.py
  -- frcnn.py
```

- **networks**: 关于faster-rcnn网络的代码
  - **model_data:** 存放模型权重，例如预训练模型、训练的faster-rcnn。
  - **resnet50.py**: 作为faster-rcnn的backbone，用于特征提取
  - **rpn.py**: RPN网络构建的代码
  - **classifier.py:** 检测头的代码
  - **frcnn.py:** faster-rcnn的整体框架代码
  - **frcnn_training.py:** faster-rcnn训练需要的配置，例如优化器，学习率等
- **utils**:
  - **anchors**: 基础先验框和每个网络拓展先验框的代码

    - **utils_bbox**: 预测框与先验框融合以及解析的代码
    - **dataloader.py:** 生成一个关于VOC2007的dataloader类，用于训练或验证。
    - **utilss:** 存放一些常用的函数，如随机数种子设置，图像归一化等
    - **utils_map.py:** 计算map的库
    - **utils_fit.py:** 训练时，单个epoch的过程
    - **callbacks.py:** 训练过程，记录数据
- **dataset**:存放了VOC2007数据集,其中:
    - **Annotations**: ".xml"格式的文件，记录每张图片的大小、物体类别、边界框位置信息等
    - **ImageSets**: 关于数据集划分成train、val、test的".txt"文件
    - **JPEGImages**: JPEG格式的图片
    - **SegmentationClass**: 以类别为基准的分割图片,可做语义分割
    - **SegmentationObject**: 以物体为基准的分割图片,可做实例分割
- **train.py:** 利用VOC2007训练faster-rcnn的训练框架
- **voc_annotation.py:** 将VOC数据集做注释处理，生成的".txt"文件，将带有图像路径，以及真实框和标签信息。
- **get_map.py:** 计算map
- **predicet.py:** 模型预测文件
- **summary.py:** 计算模型的网络结构，包括参数量，以及FLOPS等
- **frcnn.py:** 模型预测调用的库，主要是方面实现预测的4个功能

2. faster-rcnn网络框架整体介绍



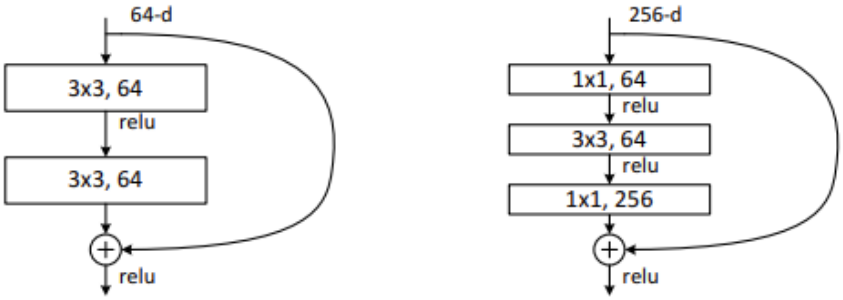


- 步骤：
  - **第一步**:将整个图片输入到一个基础卷积网络，得到整张图的feature map；
  - **第二步**:将region proposal（ROI）映射到feature map当中；
  - **第三步**:ROI提取一个**固定长度**的特征向量，每个特征会输入到一系列全连接层，得到一个ROI特征向量（**该步骤对每一个候选区域都会进行同样的操作**）；
    - 其中一个是传统softmax层进行分类，输出类别有K个类别+“背景”类；
    - 另一个是bounding box regressor。

2.1. backbone构建

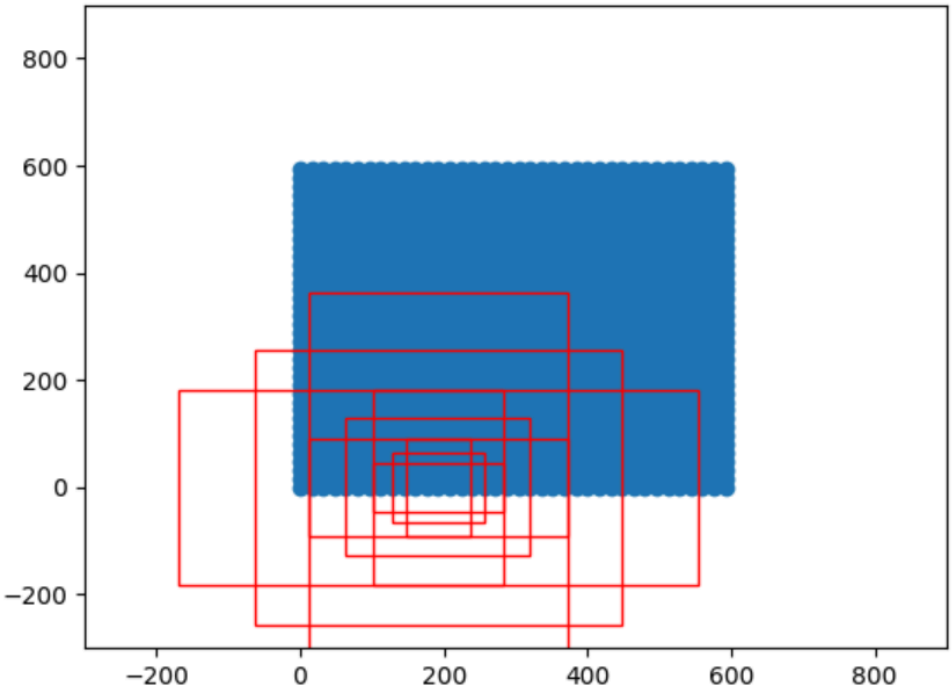
layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

这里选用的是resnet50的结构, 为了能够顺利加载resnet50网络权重，因此，严格按照pytorch官方给的命名结构来写。



其中注意的是残差块的大小，**左图是resnet18/34的残差块，右图是resnet50/101/152的残差块，这里如果当模型需要进行高和宽的压缩的时候，一般右图会有downsample操作。**

2.2. RPN构建



这里的RPN其实是建立在原图的基础上，在最后分割头的时候，会根据比例缩放到feature map上，所以为什么就有了大约在600*600的网格点出来。 每个网格点，都会得到9个先验框，组成一共12996个先验框，根据NMS在训练过程中，筛选出600个先验框出来。

这里讲解的文件是**rpn.py**：

- **1**.先在实例化的时候，就初始化9个**基础先验框**。
- **2**.从forwad()开始看，首先是根据backbone的feature map，经过简单的卷积操作，得到两条分支，分别是**rpn_locs**和**rpn_scores**，用于记录**预测框的位置**和**对应的得分**。与此同时，根据得分，筛选出有物体的预测框出来做准备。值得提及的是根据后来的代码查阅，发现，其实**rpn_locs**的最后一维4个参数分别代表(**左上x坐标，左上y坐标，以左上x为基准的W，以左上y为基准的H**)，所谓的

预测框，其实更像是在先验框的基础上，通过参数化的微调从而实现的。**这里可以看出双端目标检测网络，其实是需要先验框的配合，结合预测数值搭配完成预测。**

- **3**.**_enumerate_shifted_anchor(np.array(self.anchor_base), self.feat_stride, h, w)**：将生成的**基础先验框**根据**self.feat_stride**参数，进行平移调整，覆盖到每一个网格中心。返回得到一个每个网格点的9个预测框，以这里为例shape就是(12996, 4)
- **4**.接下来，需要将先验框们根据batch覆盖在每一张图片里。**self.proposal_layer(rpn_locs[i], rpn_fg_scores[i], anchor, img_size, scale=scale)** 完成了12996个先验框与预测点的匹配问题，**根据NMS算法，以及其nms_iou、xxx_post_nms**参数，最后筛选出一系列得分较高的带有预测性质的先验框，此时可以称之为**预测框**。模式是训练的时候，shape为(600, 4)，大大减少了预测框数量。
- **5**.最后根据**torch.cat**操作，将一个batch的预测框都合在一起。返回5个参数
  - **rpn_locs：** 每个样本筛选后的600个预测框
  - **rpn_scores：** 每个预测框对应的得分，或者也叫置信度。
  - **rois：** 存放一个批次的所有筛选后的roi。
  - **roi_indices：** 记录roi属于哪个批次的，同一批次的indice相同。
  - **anchor：** 未筛选前的12996个基础先验框。

2.3. 检测头

因为backbone是resnet50，所以这里是以此为基础的检测头，这里讲解的文件是**classifier.py**。

步骤：

- **1.** 先将batch与每个类别预测框数**合并**后，展平。然后，**将ROIs的高度坐标从图像坐标系转换为特征图坐标系**。
- **2.** 将indices与rois匹配合并起来，经过**ROIPooling**，得到shape:**(batch * 600, 1024, 14, 14)**。
- **3.** 利用resnet的第五层conv作为分类提取，将roipooling的输出送入sel.classifier，得到fc7，其shape:**(batch * 600, 2048, 1, 1)**，view后，得到 **(batch * 600, 2048)**。
- **4.** fc7分别送到self.cls_loc和self.score，得到 **(batch * 600, n_class*4)** 和 **(batch * 600, n_class)**，其中roi_cls_locs相当于对所有的类别都进行了预测框预测，roi_scores记录了每个预测框的得分情况。
- **5.** 最后view尺寸，将batch与每个类别预测框数**分离**，得到最终输出，得到 **(batch, 600, n_class*4)** 和 **(batch, 600, n_class)**

可以看到，变量rois与roi_indices并未经过分割头，这也说明了RPN重在与调整锚框，而分割头重在于评价锚框，并根据此返回给RPN。

3. VOC数据集预处理

该预处理文件就是**voc_annotation.py**，这里的2007_train.txt，2007_val.txt文件里面，不仅有样本对应的文件名，还有类别、bbox标签的信息，后续的训练主要也是使用这个

annotation_mode 具体说明如下：

- annotation_mode == 0: 根据Annotations文件夹以及ImageSets/main里的样本数据生成trainval.txt, train.txt, val.txt, test.txt, 2007_train.txt，2007_val.txt文件
- annotation_mode == 1: 根据Annotations文件夹里的样本数据生成trainval.txt, train.txt, val.txt, test.txt
- annotation_mode == 2 **（推荐直接选择这个模式）**：根据ImageSets/main文件夹里的样本数据生成2007_train.txt，2007_val.txt文件
  这里的2007_train.txt，2007_val.txt，是从里面获取的。

以2007_train.txt为例，其每一行的内容：（图片名字，边界框1，边界框1的类别，边界框2，边界框2的类别, ...,）

4. dataloder的讲解

该文件在utils/dataloder.py，这一步与voc_annotation.py衔接紧密

具体说明如下：

- 该dataloader将返回**image, box:(左上点x坐标, 左上点y坐标, 右下点x坐标, 右下点y坐标), label**，这里的数据是以两个点的坐标为标签的，**但是在RPN网络里面的预测，其实是通过左上点，以及W,H做的修正，并不冲突**，因为两点相减就能得到W,H。
- 该dataloader将根据要求的尺寸，先布局一张灰色背景，然后将原图和真实框放在灰色背景正中央。
- 这里其实我调整了原公开的代码，没有在训练的时候采取缩放处理。因为缩放后，会导致真实框与实际缩放图片内容不符合，后续再做改进吧。
- **frcnn_dataset_collate**函数的作用是在训练时，将每个batch的数据**从numpy转为torch。**

5. 训练过程train.py的讲解

如果没有特殊说明，都是在讲解**train.py**，随着代码解释越深，就需要引出其他文件，届时会指出。

整个网络训练，有冻结和解冻两个阶段，具体流程如下：

- 1.超参数配置。

- 2.加载模型。
- 3.Loss记录的初始化。
- 4.数据集加载。
- 5.迭代次数的判断。
- 6.学习率和优化器的设置。
- 7.初始化FasterRCNNTrainer类。
- 8.记录eval的map曲线。
- 9.模型训练的过程--冻结训练和解冻训练。
- 10.每次训练轮回的函数讲解。

5.1. 超参数配置

1. Cuda[bool]: 判断是否使用GPU
2. seed[int]: 设置所有需要的随机数种子，这样能够保证下每次训练获得结果一样
3. train_gpu[list]: 训练所用到的GPU数量号
4. fp16[bool]: 是否使用混合精度训练
5. classes_path[str]: 对应训练数据集的标签类别名称的存放路径
6. model_path[str]: 整个模型的权重路径，这里不影响backbone的预训练权重
7. input_shape[list]: 网络输入尺寸，也是对数据集中图片输入大小的设定
8. backbone[str]: 设置backbone的名称，例如vgg和resnet50
9. pretrained[bool]: 设置backbone是否使用预训练权重
10. anchors_size[list]: 先验框的大小设置，具体可见anchors.py
11. Init_Epoch[int]: 初始训练轮数
12. Freeze_Epoch[int]: 模型冻结训练的轮数，这里冻结了backbone的权重
13. Freeze_batch_size[int]: 冻结训练时的batch
14. UnFreeze_Epoch[int]: 总训练轮数，解冻的轮数应该是UnFreeze_Epoch-Freeze_Epoch
15. Unfreeze_batch_size[int]: 模型在解冻后的batch_size
16. Freeze_Train[bool]: 是否进行冻结训练
17. Init_lr[float]: 模型的初始学习率
18. Min_lr[float]: 模型的最小学习率（训练过程中调整的最小学习率）
19. optimizer_type[str]: 优化器种类，可选的有adam、sgd
20. momentum[float]: 优化器内部使用到的momentum参数
21. weight_decay[int or float]: 权值衰减参数（防止过拟合）
22. lr_decay_type[str]: 使用到的学习率下降方式，可选的有'step'、'cos'
23. save_period[int]: 多少个epoch保存一次权值
24. save_dir[str]: 权值与日志文件保存的文件夹路径
25. eval_flag[bool]: 是否在训练时进行评估，评估对象为验证集
26. eval_period[int]: 代表多少个epoch评估一次
27. num_workers[int] 用于设置是否使用多线程读取数据
28. train_annotation_path[str]: 训练集路径，即2007_train.txt文件
29. val_annotation_path[str]: 验证集路径，即2007_val.txt文件

根据超参数的名称以及配置，也能大致了解训练代码的整体框架。

5.2. 加载模型

这里可以选择是否要加载训练好的faster-rcnn权重，一般第一次训练网络是没有的。
这里原作者提供了网盘代码，有需要的可以自行去找。

这里的加载方式划重点：

```
model_dict = model.state_dict()
pretrained_dict = torch.load(model_path, map_location=device)
load_key, no_load_key, temp_dict = [], [], {}
for k, v in pretrained_dict.items():
    if k in model_dict.keys() and np.shape(model_dict[k]) == np.shape(v):
        temp_dict[k] = v
        load_key.append(k)
    else:
        no_load_key.append(k)
model_dict.update(temp_dict)
model.load_state_dict(model_dict)
```

这种方法加载了现在写的模型里，能够加载的权重键值对，而避免了不在'.pth'的键值对报错现象。
其思路则是，先将现在模型的键值对取出，同时也加载预训练的键值对，根据匹配更新现在模型的键值对，并且重新加载。

5.3. Loss记录的初始化

这里要看callbacks.py文件里面的LossHistory类，采用了两种记录方式：

- **第一种：** tensorboard中SummaryWriter类来进行Loss值的记录。
- **第二种：** txt与plot来记录loss值。

5.4 数据集加载

根据变量train_annotation_path和val_annotation_path找到
"VOC2007_train.txt"和"VOC2007_val.txt"的路径，并且使用写好的dataset类利用Dataloader加载。
注意，这里的batch size需要判断是不是冻结训练，冻结和解冻两个阶段的batch size可以单独。

```
wanted_step = 5e4 if optimizer_type == "sgd" else 1.5e4
total_step = num_train // Unfreeze_batch_size * UnFreeze_Epoch
```

这里的要求 total_step >= wanted_step， 如果不满足就建议调整一下，这里即便不调整，也会给出**建议的训练轮次设置**。

5.5 迭代次数的判断

这里主要是判断总的迭代次数（epoch乘以每一次epoch需要迭代的次数）：模型训练的迭代次数（由数据集规模体现）是否大于**不同优化器的要求次数**，这里的要求次数是个大概范围。

5.6 学习率和优化器的设置

根据是否采取冻结训练，确定batch_size，然后根据batch_size选择学习率。
同时，选择好优化器。

5.7 初始化FasterRCNNTrainer类

参考**frcnn_training.py**，该类主要是将model包装好，然后在训练的时候，通过调用函数的形式计算损失并返回。

在训练过程中，该类的forward具体步骤：
输入 ---> imgs, bboxes, labels, scale
输出 ---> losses

- i. 先**获得公共特征**，即backbone输出的feature map。
- ii. 根据公共特征，利用rpn网络**获得调整参数、得分、建议框、先验框**，接下来**取batch的每一个样本处理。**
- iii. 先利用**先验框**和**真实边界框**计算**rpn网络的损失函数**：
  - a. 先计算每个真实框与其所有先验框的iou值，并根据此来打上**类别标签。**这里是类别标签（1表示正样本，0表示负样本，-1则忽略），这里的正样本负样本实则就是先验框与真实框的IoU是否大于阈值判断的。**这里保证正负样本数量平衡，各为128,一共256个样本参与了损失函数计算。**
  - b. 根据每张图片的每个真实框，给每个真实框的所有先验框打上**定位标签。**定位标签，则是通过真实框和先验框的换算关系得到的定位偏移量。
  - c. 利用**定位标签计算回归损失，利用类别标签计算分类损失。**回归损失选用Smooth L1损失，分类损失选用交叉熵损失函数(这里忽略掉标签-1的计算)。
- iv. 再利用**建议框**和**真实边界框**计算**classifier网络的损失函数**：
  - a. 先计算每个真实框与其所有建议框的iou值，并根据此打上**定位标签和类别标签。**类别标签是**真实的样本标签**+1（因为有背景0）。**保证计算的正负样本数量平衡，各为56，一共128个样本参与了损失函数计算。**
  - b. 利用faster-rcnn的*head*模式，得到最终预测的定位loc预测与标签预测。

    - c. 利用**定位标签计算回归损失，利用类别标签计算分类损失。**回归损失选用Smooth L1损失，分类损失选用交叉熵损失函数(这里忽略掉标签-1的计算)。
- v. **返回一个包含五个元素的loss列表**: (RPN定位损失, RPN分类损失, ROI定位损失, ROI分类损失, 所有损失项的和)

5.8 记录eval的map曲线

请自行了解map指标知识点，这里暂时跳过！

5.9 模型训练的过程--冻结训练和解冻训练

模型在设置的总epoch，即UnFreeze_Epoch-Init_Epoch里，每一个epoch训练步骤如下：

- i. 首先判断是冻结训练的轮次还是解冻训练的轮次。如果是冻结训练的轮次，保持训练。如果是解冻训练的轮次，那就需要根据batch_size重新调整学习率，重新加载dataloader，再训练。
- ii. 根据训练轮回来更新学习率
- iii. 利用fit_one_epoch函数完成每个轮回的训练

5.10 fit_one_epoch函数步骤讲解

具体步骤：

- i. **开始训练：** 调用FasterRCNNTrainer的实例，计算损失函数，并反向传播。
- ii. **开始验证：** 调用FasterRCNNTrainer的实例，计算损失函数，不反向传播。
- iii. **将损失函数记录下来。**
- iv. **计算map。**
- v. **保存模型权重：** 第一，按照epoch的周期形式存放；第二，按照验证损失最好的模型存放；第三，存放当前epoch模型权重。

6. predict.py的讲解

预测文件一共可以实现4种模式的推理：

- **predict：** 表示单张图片预测。
- **video：** 表示视频检测。
- **fps：** 表示测试fps。
- **dir_predict：** 表示遍历文件夹进行检测并保存。

6.1 超参数配置

1. model[str]: 4种推理模式
2. crop[bool]: 在predict模式下，指定是否在单张图片预测后对目标进行截取
3. count[bool]: 在predict模式下，指定是否进行目标的计数
4. video_path[int, str]: 在video模式下，待检测视频的路径，可以设置为0调用摄像头
5. video_save_path[str]: 在video模式下，保存检测视频的路径
6. video_fps[float]: 在video模式下，用于保存的视频的fps
7. test_interval[int]: 在fps模式下，指定测量fps的时候，图片检测的次数
8. fps_image_path[str]: 在fps模式下，指定测试图片路径
9. dir_origin_path[str]: 在dir_predict模式下，检测的图片的文件夹路径
10. dir_save_path[str]: 在dir_predict模式下，检测完图片的保存路径

预测功能的4种模式实现是调用**predict.py**同级下的**frcnn.py**中的FRCNN类当中方法实现的接下来讲解该类即可。

6.1 预测需要的FRCNN类

事实上faster-rcnn的目标检测结果是通过：rpn网络得到的roi与最后classifier回归出的预测调整参数结合，得到的目标预测结果。

第一步，初始化FRCNN类
这里需要完成动态属性设置、获得种类和先验框的数量、 边界框解码工具、画框设置不同的颜色以及载入模型等初始化操作。
这里的边界框解码工具需要参考**utils_bbox.py**里的DecodeBox类，6.2小节会讲解。

第二步，detect_image方法
接收一张图像作为输入，通过模型进行目标检测，并将检测结果绘制在图像上。

- 计算输入图片的高和宽
- 计算resize后的图片的大小
- 图像预处理
- 模型推理
- 提取检测结果
- 设置字体与边框厚度
- 目标计数
- 目标裁剪（如果crop为True）
- 图像绘制

第三步，get_FPS方法
返回预测的fps数值

第四步，get_map_txt方法
返回map计算值

6.2 DecodeBox类

DecodeBox类主要功能是将目标检测模型的输出（如候选框、类别置信度等）解码为最终的检测结果。因为需要与rpn网络的**roi**以及classifier回归预测的**调整参数**进行**结合**。

1.frcnn_correct_boxes方法
将框的中心点坐标和宽高转换为左上角和右下角坐标，将框的坐标从输入图像的尺度调整到原始图像的尺度。

2.重写forward方法

- **回归参数调整**：将回归参数应用于候选框，调整框的坐标。
- **框的归一化**：将框的坐标归一化到 [0, 1] 范围。
- **分类置信度计算**：对类别置信度进行 softmax 操作。
- **置信度筛选**：筛选出置信度大于阈值的框。
- **非极大抑制（NMS）**：去除重叠较高的框，保留置信度最高的框。
- **框的尺度调整**：将框的坐标从归一化尺度调整到原始图像尺度。

最后返回的格式为(y_min, x_min, y_max, x_max, confidence, class_id)