从0到1 实现YOLO v3 （Part one）

原创 DoubleV0203 机器学习算法工程师 2018-06-11

译者：刘威威
编辑：黄俊嘉

本文编译自：

- https://blog.paperspace.com/how-to-implement-a-yolo-object-detector-in-pytorch/
- https://blog.paperspace.com/how-to-implement-a-yolo-v3-object-detector-from-scratch-in-pytorch-part-2/

原文作者：

- Ayoosh Kathuria
  (https://blog.paperspace.com/author/ayoosh/)

前 言

如果说非要提供一个方法快速掌握目标检测的深度学习算法的话，那就是自己从无到有的实现它，在这期间，可以对整个算法有更清晰的认识，此次系列文章旨在提供一个自己从无到有实现目标检测YOLOV3的教程，希望对那些对目标检测感兴趣的人有所帮助。

目标检测很大程度上依赖于深度学习技术的发展，比如YOLO，SSD，MaskRCNN 和RetinaNet.

本文将详细介绍如何使用Pytorch从0到1完成YOLO v3算法，实现基于python3.5，Pytorch3.0，文中提到的所有代码都可以从Github中找到。
教程包括五个部分，本文只涉及第一个部分：

- 第一部分：理解YOLO的工作原理
- 第二部分：建立神经网络层
- 第三部分：完成整个网络的搭建
- 第四部分：目标score阈值化和NMS（非极大值抑制）
- 第五部分：完成整个网络的从输入搭配输出流程

01

准备工作

本文假设读者已经具备一下几点要求：
首先：你应该已经理解了CNN的工作原理，还有残差结构，跨层连接和上采样操作。

其次：要对目标检测的基础知识有所了解。比如bounding box regression, IoU 和 non-maximum suppression.
再者：可以熟练的使用pytorch搭建神经网络

对于没有达到上述要求的同学，建议先阅读以下链接内容了解相关知识。

- YOLOv1：https://arxiv.org/pdf/1506.02640.pdf
- YOLOv2：https://arxiv.org/pdf/1612.08242.pdf
- YOLOv3：https://pjreddie.com/media/files/papers/YOLOv3.pdf
- CNN：http://cs231n.github.io/convolutional-networks/
- bounding box regression：https://arxiv.org/pdf/1311.2524.pdf
- IoU ：https://www.youtube.com/watch?v=DNEm4fJ-rto
- non-maximum suppression：https://www.youtube.com/watch?v=A46HZGR5fMw
- PyTorch Official Tutorial: http://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html

理解YOLO

YOLO的全名是：You Only Look Once. YOLO是一个基于DeepCNN的目标检测算法，
由75个卷积层组成，其中使用卷积代替池化进行下采样，防止pooling带来的信息丢失问题。由于是全卷积结构，YOLO对输入的图像大小没有限制，但是实际中当我们刚开始实现这个算法的时候，还是先把输入图像size固定到一个大小进行训练和测试。

网络的下采样倍数通过stride参数限制，假设网络的输入大小是416x416，网络总stride是32的话，那么最终的输出就是416/32=13*13。stride表示一个输出比输入小多少倍的因子（卷积而不是转置卷积中）。

目标检测算法基本都是一个套路，首先是深度卷积网络提取特征，后接一个分类器和回归器，预测图像的label和bounding box的坐标。

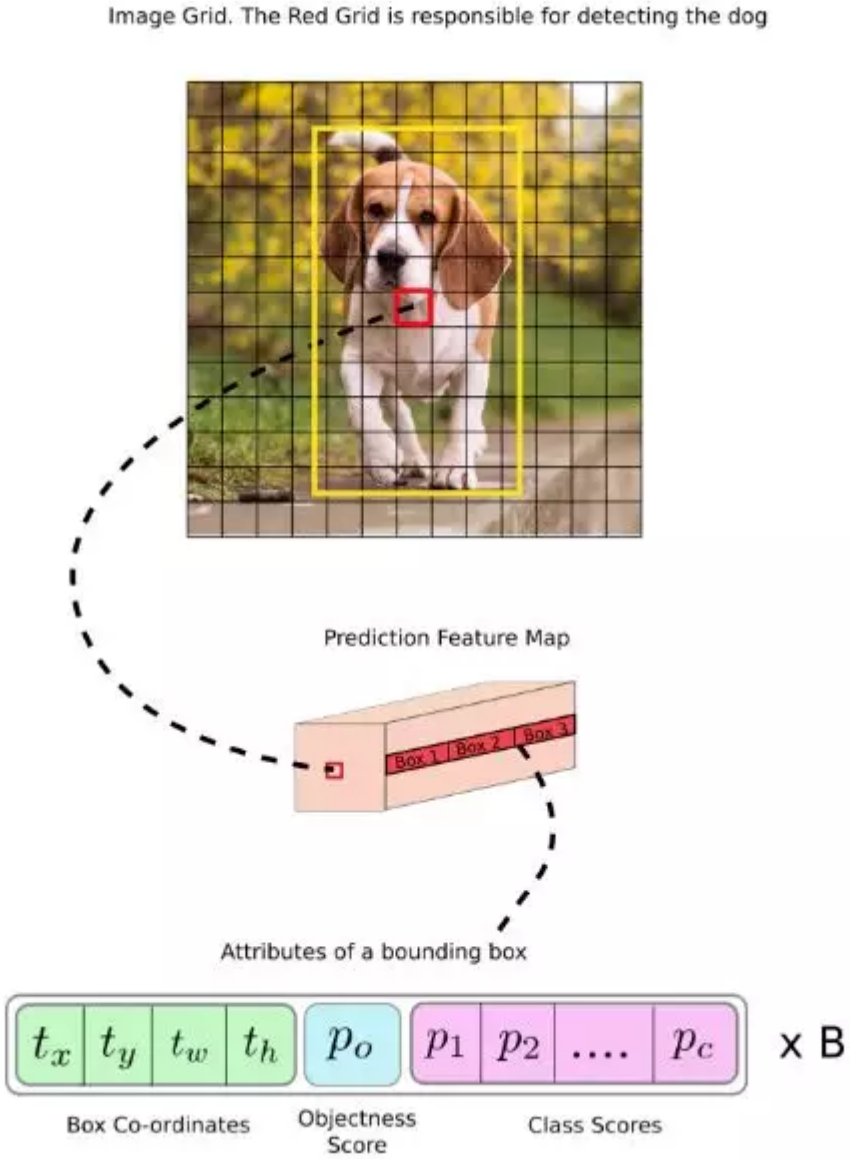
在YOLO算法中，由于他是全卷积网络结构，所以最终的预测也是用卷积网络完成，卷积核的大小是1x1x(Bx(5+C))，YOLOv3中，最终的输出是一个feature map，最终的预测feature map的大小和前一层的大小是一样的，可以把最后的featuremap理解为predict map，predict map中每一个单元可以预测固定数量的bounding box，比如预测3个。深度上，有(Bx(5+C)) 个通道，B表示每个单元可以预测的boundingbox数量，从paper中得知，每一个bounding box预测一个目标，包含有5+C个属性，包括bounding box的中心坐标，长宽，目标得分，和C类的分类置信度。

如果目标的中心位于该单元格的感受野中，则希望特征映射的每个单元格都可以通过其中一个边界框来预测对象。 （感受区域是输入图像对单元可见的区域，参见卷积神经网络链接以进一步说明）。

这与YOLO如何训练有关，只有一个边界框负责检测任何给定的对象。 首先，我们必须确定这个边界框属于哪个单元格。

为此，我们将输入图像划分成与最终特征映射相等的维度网格。

让我们考虑下面的一个例子，其中输入图像是416 x 416，网络的stride是32。如前所述，特征映射的维度将是13 x 13.然后，我们将输入图像划分为13 x 13 的单元格。



然后，选择包含对象的的单元格（在输入图像上红色格子）负责预测对象该对象（狗）。 在图像中，其包含了真实的bounding box的中心坐标（真实的bounding box标记为黄色）。

现在，红色单元格是网格中第7行的第7个单元格。 我们现在将第7行中的第7个单元分配到特征映射（特征映射中的相应单元）上作为负责检测狗的单元。

现在，这个单元格可以预测三个bounding box，最终只选择一个作为最终的预测。

请注意，我们在这里讨论的单元是预测feature map上的单元。 我们将输入图像分成一个网格，以确定预测特征图的哪个单元负责预测那个网格。

下面对YOLO的输出分别做介绍

03

Anchor Boxes

预测边界框的宽度和高度可能是有意义的，但实际上，这会导致训练期间不稳定的梯度。 相反，大多数目标检测器预测对数空间变换后的宽和高，或简单地偏移到预定义的默认边界框，也即是anchor。

然后，将这些变换应用于anchor box以获得预测。 YOLO v3有三个anchor，可以预测每个单元的三个边界框。对于检测上图中的狗来讲，选择和groundtruth bounding box的IOU最大的预测bounding box作为预测结果。

04

中心点预测

通常情况下，YOLO不预测边界框中心的绝对坐标。它预测的是偏移量，预测的结果通过一个sigmoid函数，迫使输出的值在0和1之间。例如，考虑上图中狗的情况。如果对中心的预测是（0.4,0.7），那么这意味着中心位于13×13特征地图上的（6.4,6.7）。 （因为红细胞的左上角坐标是（6,6））。

但如果预测的x，y坐标大于1，会发生什么情况，比如（1.2,0.7）。这意味着中心位于（7.2,6.7）。注意现在中心位于我们的红色区域或第7排的第8个单元格的右侧。这打破了YOLO背后的理论，因为如果我们假设红色区域负责预测狗，狗的中心必须位于红色区域中，而不是位于红色区域旁边的其他网格里。

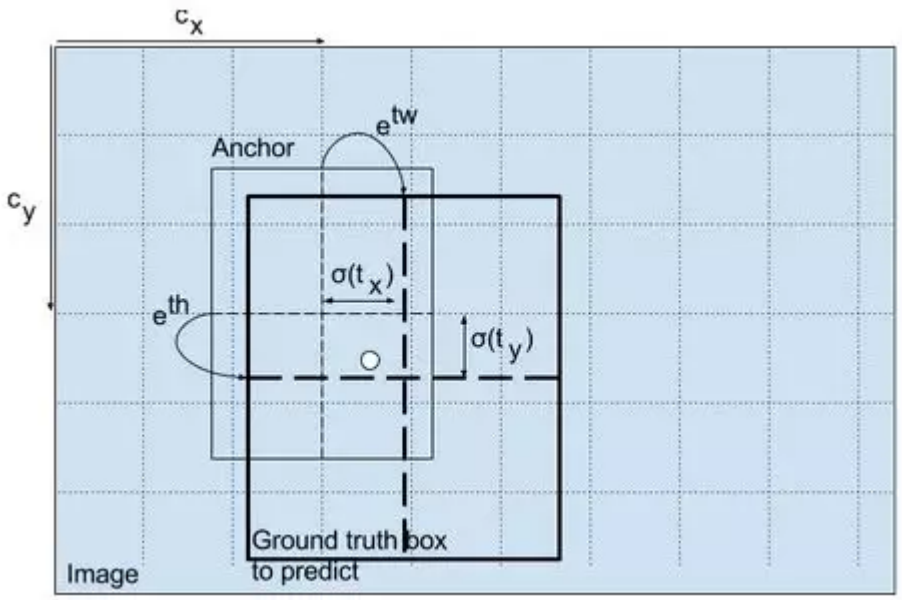
因此，为了解决这个问题，输出是通过一个sigmoid函数传递的，该函数在0到1的范围内压扁输出，有效地将中心保持在预测的网格中。下面的这个公式详细展示了预测结果如何转化为最终box的预测的。

$$
\begin{aligned}
b_x &= \sigma(t_x) + c_x \\
b_y &= \sigma(t_y) + c_y \\
b_w &= p_w e^{t_w} \\
b_h &= p_h e^{t_h}
\end{aligned}
$$

05

边界框的尺寸预测

通过对输出应用对数空间变换，然后与anchor相乘来预测边界框的尺寸。



得到的预测bw和bh通过图像的高度和宽度进行归一化。（训练标签是这样的）。 因此，如果包含狗的盒子的预测值bx和by为0.3,0.8，则13 x 13特征映射上的实际宽度和高度为（13 x 0.3,13 x 0.8）。

06

目标得分预测

目标分数表示目标包含在bounding box内的概率（对象分数也通过sigmiod函数归一化）。 红色和相邻网格应该接近1，而角落网格接近0。

07

类别的置信度

类别置信度表示属于特定类别（狗，猫，香蕉，汽车等）的检测对象的概率。 在v3之前，YOLO曾经用softmax评分。

但是，该设计选择已经在V3中被删除，作者选择使用sigmoid。 原因是Softmax会假设预测是相互排斥的。 简而言之，如果一个对象属于一个类，那么它保证它不能属于另一个类。
但是，如果类别中有woman和preson两个类别，那么softmax最终的预测只有一个，woman或者person，但是sigmoid可以实现woman和person。 这就是作者们避免使用Softmax激活的原因。

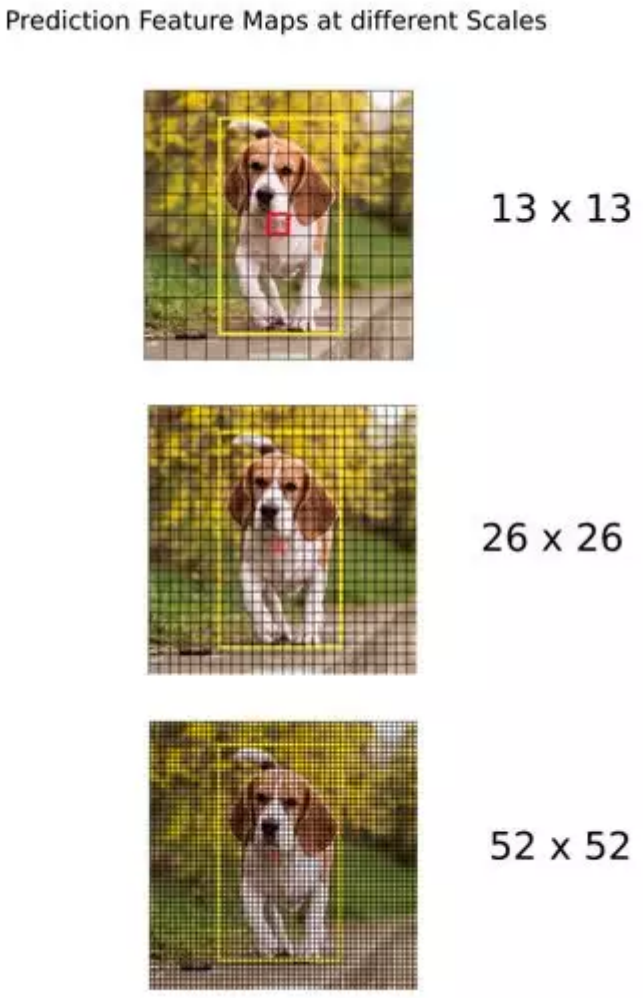
08

不同尺度的预测

YOLO v3可以进行3种不同尺度的预测。 使用检测层在三种不同尺寸的特征图上进行检测，分别具有stride=32,16,8。 这意味着，在输入416 x 416的情况下，我们使用13 x 13,26 x 26和52 x 52的比例进行检测。

网络下采样输入图像直到第一检测层，其中使用具有步幅32的层的feature map进行检测。此外，将层上采样2倍，并与具有相同特征图的先前层的特征图连接大小。 现在在具有stride=16的层上进行另一检测。重复相同的上采样过程，并且在stride=8处进行最终检测。

在每个尺度上，每个单元使用3个anchor来预测3个bounding box，从而使用9个anchor的总数（anchor在不同尺度上是不同的）



作者在论文中说名，这有助于YOLO v3更好地检测小型物体，这是对YOLO早期版本的频繁投诉。 上采样可以帮助网络学习有助于检测小物体的细粒度特征。

09

输出处理

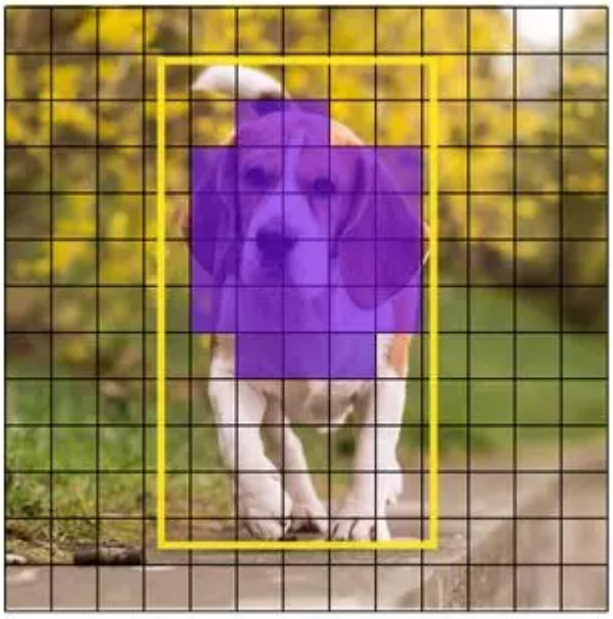
对于尺寸为416×416的图像，YOLO预测（（52×52）+（26×26）+ 13×13））×3 = 10647个边界框。 但是，在我们的形象中，只有一个物体，一只狗。 我们如何将检测结果从10647减少到1？

基于对象置信度的阈值

首先，我们根据对象分数过滤框。 通常，具有低于阈值分数的框被忽略。

非最大抑制（NMS）

NMS打算解决同一图像的多重检测问题。 例如，红色网格单元的所有3个边界框可能检测到一个对象，或者相邻单元可能检测到相同的对象。



Multiple Grids may detect the same object
NMS is used to remove multiple detections

有关YOLO的原理介绍第一部分完成了，
下面将介绍YOLO的网络搭建部分。

10

YOLO网络搭建实践

本章节主要介绍如何搭建YOLO网络，在理解本章节之前，假设你已经基本掌握了pytorch的用法。

首先，创建一个 darknet.py 文件，用来存放YOLO的网络结构部分，另建一个util.py文件，用来存放一些辅助函数。

官方提供的代码是caffe的，网络定义在.protxt里面，在这里，我们也将使用官方的cfg配置文件定义网络结构，可以从这里下载：https://github.com/pjreddie/darknet/blob/master/cfg/yolov3.cfg

把文件下载下来，然后放到代码目录里面，如果使用的是linux系统，可以直接通过如下命令完成：

```
mkdir  cfg
cd  cfg
wget https://raw.githubusercontent.com/pjreddie/darknet/master/cfg/yolov3.cfg
```

配置文件里卷积定义如下：

```
[convolutional]
batch_normalize=1
filters=64
size=3
stride=1
pad=1
activation=leaky
```

shortcut部分：

```
[shortcut]
from=-3
activation=linear
```

shortcut部分是卷积的跨层连接，就像Resnet中使用的一样，参数from是-3，意思是shortcut的输出是通过与先前的倒数第三层网络相加而得到。

上采样：

```
[upsample]
stride=2
```

Route：

```
[route]
layers = -4

[route]
layers = -1, 61
```

route层值得一些解释。 它具有可以具有一个或两个值的属性层。
当属性只有一个值时，它会输出由该值索引的网络层的特征图。 在我们的示例中，它是-4，因此这个层将从Route层向后输出第4层的特征图。

当图层有两个值时，它会返回由其值所索引的图层的连接特征图。 在我们的例子中，它是-1,61，并且该图层将输出来自上一层（-1）和第61层的特征图，并沿深度的维度连接。

YOLO:

```
[yolo]
mask = 0,1,2
anchors = 10,13,  16,30,  33,23,  30,61,  62,45,  59,119,  116,90,  156,198,  373,326
classes=80
num=9
jitter=.3
ignore_thresh = .5
truth_thresh = 1
random=1
```

YOLO层对应于第1部分中描述的检测层(detection layer)。anchor描述9个anchor，但只有mask标记的属性索引到的anchor才会被使用到。 这里，mask的值是0,1,2，这意味着使用第一，第二和第三个anchor。 这是有道理的，因为检测层的每个单元预测3个box。 总共有三个检测层，共计9个anchor，第一部分中已经提到。

Net:

```
[net]
# Testing
batch=1
subdivisions=1
# Training
# batch=64
# subdivisions=16
width= 320
height = 320
channels=3
momentum=0.9
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1
```

cfg中有另一种称为net的块，但我不会将其称为layer，因为它仅描述有关网络输入和训练参数的信息。 它不用于YOLO的正向传播。 然而，它确实为我们提供了像网络输入大小这样的信息，我们用它来调整正向传播中的anchor。

正式开始写代码前，先import一些必要的库：

```
from __future__ import division

import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.autograd import Variable
import numpy as np
```

定义一个parse_cfg函数，输入信息是cfg的文件路径，输出一个包含文件中block的字典，代码如下：

```
def parse_cfg(cfgfile):
    """
    Takes a configuration file
    Returns a list of blocks. Each blocks describes a block in the neural
    network to be built. Block is represented as a dictionary in the list

    """
    file = open(cfgfile, 'r')
    lines = file.read().split('\n')      # store the lines in a list
    lines = [x for x in lines if len(x) > 0] # get read of the empty lines
    lines = [x for x in lines if x[0] != '#'] # get rid of comments
    lines = [x.rstrip().lstrip() for x in lines] # get rid of fringe whitespaces
    block = {}
    blocks = []

    for line in lines:
       if line[0] == "[":                    # This marks the start of a new block
          if len(block) != 0:              # If block is not empty, implies it is storing values o
              blocks.append(block)       # add it the blocks list
              block = {}                    # re-init the block
              block["type"] = line[1:-1].rstrip()
          else:
             key,value = line.split("=")
             block[key.rstrip()] = value.lstrip()
     blocks.append(block)

  return blocks
```

然后就有了YOLO的网络信息，存放在blocks 的list中，就可以去搭建网络了。
list中包含了之前提到的5种类型的网络层，pytorch提供的有预定义好的convolution和upsample，只需要写我们自己的modules就可以了。

接下来定义一个create_modules函数，以上个函数输出的blocks list作为输入，对其中的信息解码并转换为pytorch的网络层：

```
def create_modules(blocks):
    net_info = blocks[0]    #Captures the information about the input and pre-processing
    module_list = nn.ModuleList()
    prev_filters = 3
    output_filters = []
    for index, x in enumerate(blocks[1:]):
        module = nn.Sequential()

        #check the type of block
        #create a new module for the block
        #append to module_list
        if (x["type"] == "convolutional"):
            #Get the info about the layer
            activation = x["activation"]
            try:
                batch_normalize = int(x["batch_normalize"])
                bias = False
            except:
                batch_normalize = 0
                bias = True

            filters= int(x["filters"])
            padding = int(x["pad"])
            kernel_size = int(x["size"])
            stride = int(x["stride"])

            if padding:
                pad = (kernel_size - 1) // 2
            else:
                pad = 0

            #Add the convolutional layer
            conv = nn.Conv2d(prev_filters, filters, kernel_size, stride, pad, bias = bias)
            module.add_module("conv_{0}".format(index), conv)

            #Add the Batch Norm Layer
            if batch_normalize:
                bn = nn.BatchNorm2d(filters)
                module.add_module("batch_norm_{0}".format(index), bn)

            #Check the activation.
            #It is either Linear or a Leaky ReLU for YOLO
            if activation == "leaky":
                activn = nn.LeakyReLU(0.1, inplace = True)
                module.add_module("leaky_{0}".format(index), activn)
```

```
        #If it's an upsampling layer
        #We use Bilinear2dUpsampling
        elif (x["type"] == "upsample"):
              stride = int(x["stride"])
              upsample = nn.Upsample(scale_factor = 2, mode = "bilinear")
              module.add_module("upsample_{}".format(index), upsample)
#If it is a route layer
        elif (x["type"] == "route"):
              x["layers"] = x["layers"].split(',')
              #Start  of a route
              start = int(x["layers"][0])
              #end, if there exists one.
              try:
                    end = int(x["layers"][1])
              except:
                    end = 0
              #Positive anotation
              if start > 0:
                    start = start - index
              if end > 0:
                    end = end - index
              route = EmptyLayer()
              module.add_module("route_{0}".format(index), route)
              if end < 0:
                    filters = output_filters[index + start] + output_filters[index + end]
              else:
                    filters= output_filters[index + start]

        #shortcut corresponds to skip connection
        elif x["type"] == "shortcut":
              shortcut = EmptyLayer()
              module.add_module("shortcut_{}".format(index), shortcut)
        #Yolo is the detection layer
        elif x["type"] == "yolo":
              mask = x["mask"].split(",")
              mask = [int(x) for x in mask]

              anchors = x["anchors"].split(",")
              anchors = [int(a) for a in anchors]
              anchors = [(anchors[i], anchors[i+1]) for i in range(0, len(anchors),2)]
              anchors = [anchors[i] for i in mask]

              detection = DetectionLayer(anchors)
              module.add_module("Detection_{}".format(index), detection)

        return (net_info, module_list)
```

OK，此部分完成了网络的搭建部分，为了验证网络搭建的是否正确，可以适用下面的代码运行，测试一下：

```
blocks = parse_cfg("cfg/yolov3.cfg")
print(create_modules(blocks))
```

输出的结果是这样的：

```
.
.

  (9): Sequential(
      (conv_9): Conv2d (128, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (batch_norm_9): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True)
      (leaky_9): LeakyReLU(0.1, inplace)
    )
    (10): Sequential(
      (conv_10): Conv2d (64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (batch_norm_10): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True)
      (leaky_10): LeakyReLU(0.1, inplace)
    )
    (11): Sequential(
      (shortcut_11): EmptyLayer(
      )
    )
.
.
```

到此，网络搭建部分已经完成，在下一次的文章中会介绍网络的训练部分，输入一张图像并输出结果。

参考文献：

https://blog.paperspace.com/how-to-implement-a-yolo-object-detector-in-pytorch/
https://blog.paperspace.com/how-to-implement-a-yolo-v3-object-detector-from-scratch-in-pytorch-part-2/
http://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html
https://arxiv.org/pdf/1506.02640.pdf
https://arxiv.org/pdf/1612.08242.pdf
https://pjreddie.com/media/files/papers/YOLOv3.pdf
http://cs231n.github.io/convolutional-networks/
https://arxiv.org/pdf/1311.2524.pdf
https://www.youtube.com/watch?v=DNEm4fJ-rto
https://www.youtube.com/watch?v=A46HZGR5fMw

END

往期回顾之作者刘威威

【1】YOLO算法的原理与实现[作者：叶虎]
【2】基础|详解机器学习中的梯度消失、爆炸原因及其解决方法
【3】免费使用谷歌GPU资源训练自己的深度模型
【4】风格迁移原理及tensorflow实现-附代码
【5】手把手教你搭建目标检测器-附代码