

# 学习记录 yolo3

2020年6月1日 15:42

Yolov3

输入: 416\*416 3通道彩色图像 像素格式: (待定)  
Convolutional 64 3x3/2 128x128

/2: 下采样 or stride

研究: 非1步长卷积的尺寸计算公式: = 从特征图每两个像素点中去掉一个~ =下采样取左边 尺寸比下采样+1  
信息低于下采样平均或最大

4 Shortcut Layer: 1

是: 跨层连接 此处做: 直接11相加

可以: concat (size扩展 通道要求一致, size某—维一致) 通道堆叠 ( size要求一致) 直接相加 (size和通道要求一致)

非: 1像素减通道层

调查

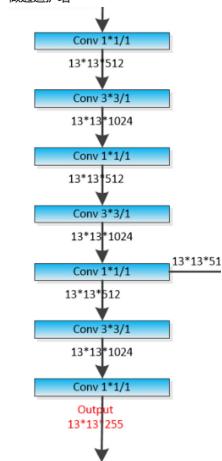
基本块格式

12 conv 256 3 x 3 / 2 104 x 104 x 128 -> 52 x 52 x 256 1.595 BF
13 conv 128 1 x 1 / 1 52 x 52 x 256 -> 52 x 52 x 128 0.177 BF
14 conv 256 3 x 3 / 1 52 x 52 x 128 -> 52 x 52 x 256 1.595 BF
15 Shortcut Layer: 12

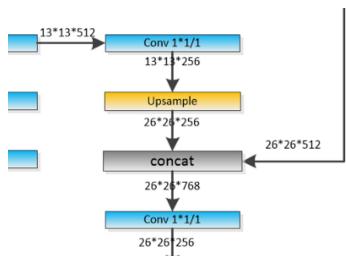
第一层存在前置层:

0 conv 32 3 x 3 / 1 416 x 416 x 3 -> 416 x 416 x 32 0.299 BF
1 conv 64 3 x 3 / 2 416 x 416 x 32 -> 208 x 208 x 64 1.595 BF
2 conv 32 1 x 1 / 1 208 x 208 x 64 -> 208 x 208 x 32 0.177 BF
3 conv 64 3 x 3 / 1 208 x 208 x 32 -> 208 x 208 x 64 1.595 BF
4 Shortcut Layer: 1

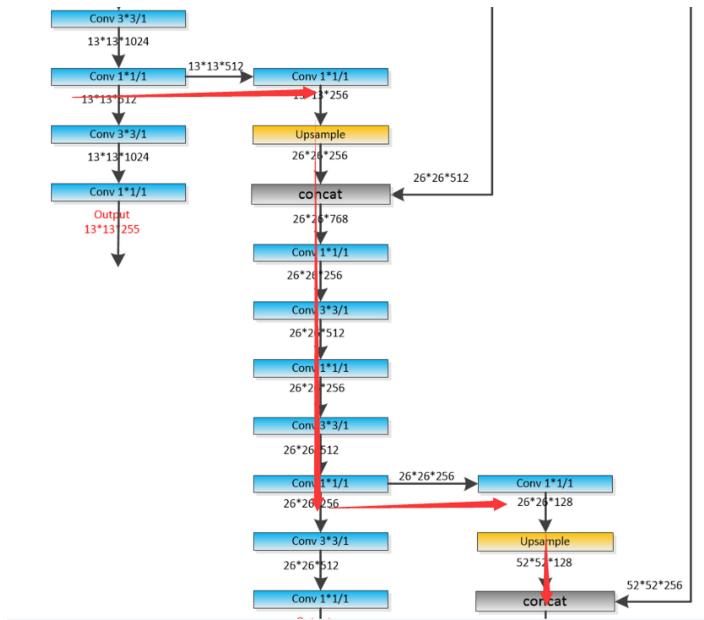
做通道扩增



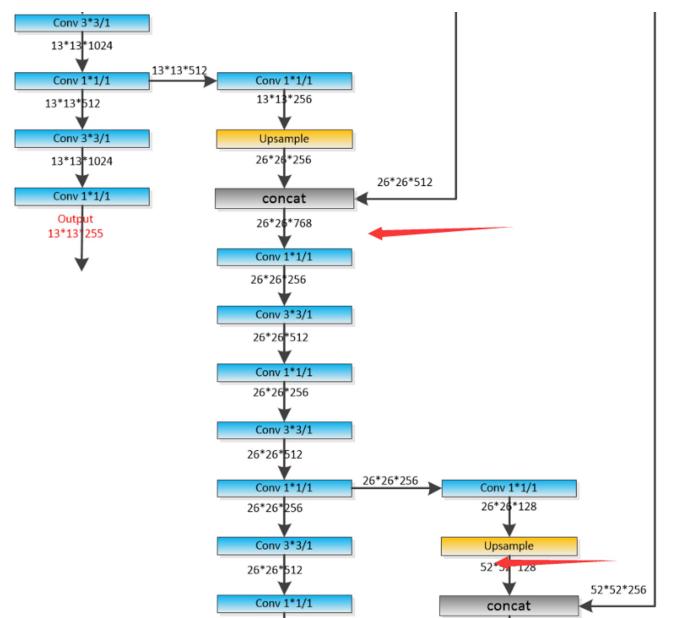
堆叠输出部分



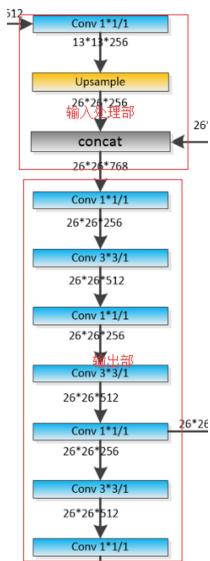
分支输入 双来源



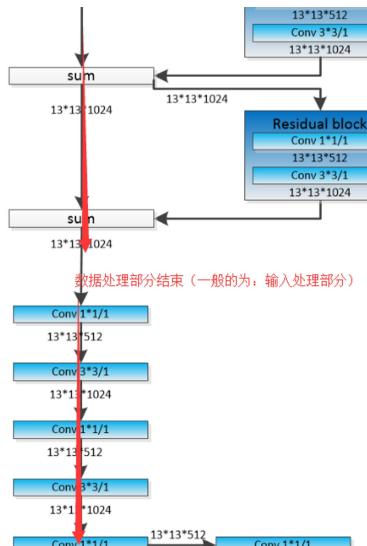
扩增流，由上级网络中部导出，经过多次扩增与转换



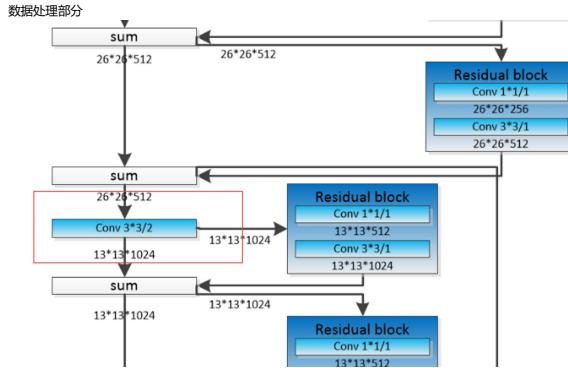
原始流，从主网络上 中间直接导出



所有分支网络的部分，其中主网络上层的输入处理部分换成残差堆叠形成的数据处理部分

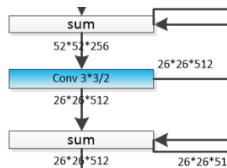


主网络前面为直接针对输入数据的数据处理部分  
主网络的双输入合并为单输入，直接处理输入数据，可认为是自己作为自己的上级网络



其上为SUM连接，尺寸一致

以上残差堆叠中断，产生一个直接的卷积（非残差），改变尺寸后继续通过残差堆叠提取特征



第一尺寸变换部分

数据处理网络结构  
特征提取->下采样->继续提取->下采样->继续提取->...->最终特征

共5个下采样节点

尺寸变化：

208 208 64

104 104 128

(以下导出数据)

52 52 256

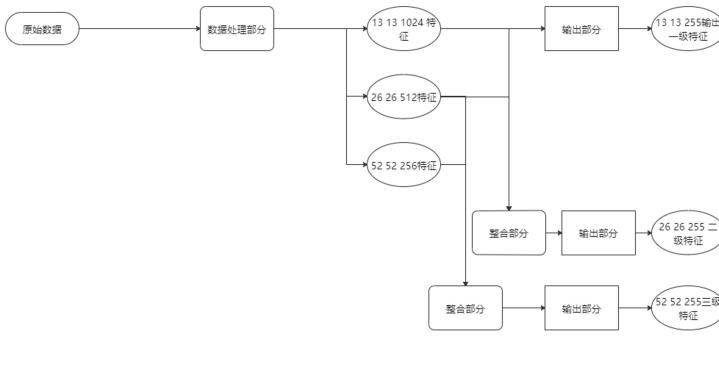
26 26 512

13 13 1024

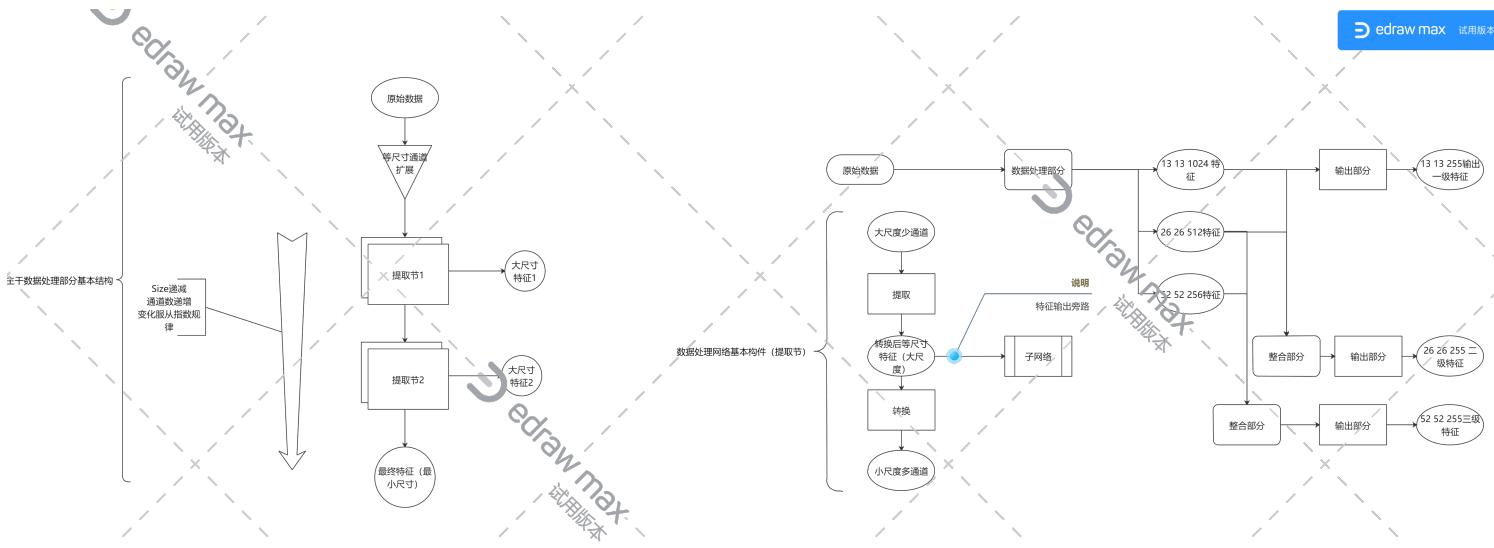
整体结构

子结构(输出组件) 前部->输出->特征数据

(准备使用箭头图表示网络结构)



整体结构



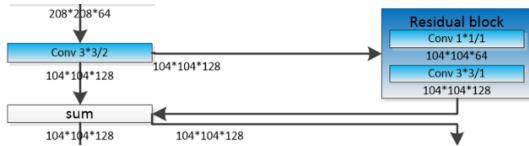
其中的残差块有几种类型

1 conv	64	3 x 3 / 2	416 x 416 x 32	$\rightarrow$	208 x 208 x 64	1.595 BF
2 conv	32	1 x 1 / 1	208 x 208 x 64	$\rightarrow$	208 x 208 x 32	0.177 BF
3 conv	64	3 x 3 / 1	208 x 208 x 32	$\rightarrow$	208 x 208 x 64	1.595 BF

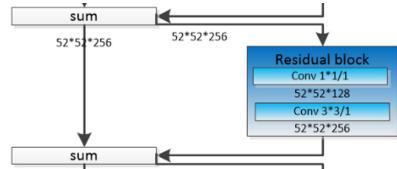
这不是一个残差块 第2层为残差块 1为shrink层

残差块的定义为 start输入与end输出shape一致 明显为2 3

图中表现为



而不跟在shrink层后面的单纯的残差块变现为:



上方sum为上一个残差块结尾 因此其表现为一个shortcut层，其下的所有conv到下一个shortcut都是一个resblock

resblock类型

- 1. 无middle型 常在前面
- 2. 有middle型

三元组定义中 hwc指一个卷积层的输出shape的参数

Triple pair中的start和end指开始层的输出 和 end层的输出

约束: end层的输出的shape should match the shape of the input of start layer

And the information of input of start layer is not in the parameters above

Should I provider the paramters of the start layer? Is it important? is the start layer the same as a

middle layer?

We all know that

结论:

在yolo3的resblock中，都为两层，第一层做通道缩减，第二层做通道恢复，shape不变

其中的卷积块的排列都是 通道缩减->扩展->缩减->扩展 而size一般不变

即使是在output block中也是如此

而残差块则是 缩减->扩展还原 两个conv

而缩减，据观察基本都是指数规律，即减半，还原，减半，还原

据观察缩通道用的是 1\*1/1的卷积 通道减半 尺寸不变

缩尺寸用的是 3\*3/2 same 的卷积 size减半 通道增加一倍

一个单位为:

尺寸变换定义为 size/2 c\*2

Size c input

Size/2 c\*2 尺寸变换 准备输入残差块

Size/2 c 缩通道

Size/2 c\*2 通道还原

输出 为 一次尺寸变换的tensor

另一个单位为 尺寸恒定的残差块

Size c input

Size/2 c/2 通道缩小

Size c output 通道还原  
这里可以命名为: shape恒等变换  
前者的size/2 c<2 可以认为是: shape变换  
前者可以看作一个shape变换加一个shape恒等变换的叠加

输出层为三个shape恒等变换堆叠, 后跟一个output层  
output层为一个1\*1/1卷积层, 通道均为255, 不改变size  
每个网络, 包括主干和子网络的输出部分, 格式一样, 如果要对夏季输出特征, 则于shape恒等变换的缩通道层  
输出时, 导出一个feature, 其通道为原始通道的二分之一

子网络的head层  
由两部分组成 扩增层->concat层, 扩增曾为扩增size, concat层以通道为堆叠, 扩增会把上一级的特征  
的尺寸拉到和上一级特征一样, 而通道只有其一半, 因此concat后, 得到的特征的尺寸与上一级特征的尺寸一  
致, 而通道增加了一半  
可以认为是沿着轴3concat concat(axis=3)  
这样

需要注意的是, shape恒等变换层和残差块不一样,  
output中是恒等变换层的线性堆叠并没有跨层链接

yolo结构记录  
Head->shrink->res1

关于数据集的问题  
可使用voc数据集  
其格式如下

```
{'image': <tf.Tensor 'IteratorGetNext_1:0' shape=(?, ?, 3) dtype=uint8>,
'image/filename': <tf.Tensor 'IteratorGetNext_1:1' shape=() dtype=string>,
'labels': <tf.Tensor 'IteratorGetNext_1:2' shape=(?,) dtype=int64>,
'labels_no_difficult': <tf.Tensor 'IteratorGetNext_1:3' shape=(?,) dtype=int64>,
'objects': {'bbox': <tf.Tensor 'IteratorGetNext_1:4' shape=(?, 4) dtype=float32>,
'is_difficult': <tf.Tensor 'IteratorGetNext_1:5' shape=(?,) dtype=bool>,
'is_truncated': <tf.Tensor 'IteratorGetNext_1:6' shape=(?,) dtype=bool>,
'label': <tf.Tensor 'IteratorGetNext_1:7' shape=(?,) dtype=int64>,
'pose': <tf.Tensor 'IteratorGetNext_1:8' shape=(?,) dtype=int64>}}
```

数据集名字: voc2007 ; in tf.ds

数据集分为: train test validation 三个

```
voc.keys()
dict_keys(['test', 'train', 'validation'])
```

```
'image': <tf.Tensor 'IteratorGetNext_1:0' shape=(?, ?, 3) dtype=uint8>,
'image/filename': <tf.Tensor 'IteratorGetNext_1:1' shape=() dtype=string>,
'---对应 image需要格式转换/255 三通道彩色图像
'objects': {'bbox': <tf.Tensor 'IteratorGetNext_1:4' shape=(?, 4) dtype=float32>,
'is_difficult': <tf.Tensor 'IteratorGetNext_1:5' shape=(?,) dtype=bool>,
'is_truncated': <tf.Tensor 'IteratorGetNext_1:6' shape=(?,) dtype=bool>,
'label': <tf.Tensor 'IteratorGetNext_1:7' shape=(?,) dtype=int64>,
'pose': <tf.Tensor 'IteratorGetNext_1:8' shape=(?,) dtype=int64>}}
```

这是一个子对象, 其中包含了bbox 大哥xyhw 4个坐标  
is\_difficult is\_truncated 都是bool的标量 意义不明 (调查需求)  
label是一个int 标量, 大概是一个idx, 表示bbox中的物体的类  
pose大概表示其姿态, 也是一个idx, 属于类标签的一种

问题是 objects是否多个?

objects外面的image 中可以有多个label 即框  
而下面的labels标量大概表示有多少个box  
labels下面的大概表示有多少个 “不困难的” box

调查: “困难”的定义 truncated的定义

调查: 这之中多个box怎么得到

```
[120, 85, 45],  
[[133, 116, 100],  
[137, 119, 99],  
[162, 138, 110],  
...,  
[182, 147, 105],  
[189, 154, 112],  
[160, 125, 83]],  
[[140, 119, 100],  
[156, 134, 111],  
[163, 138, 108],  
...,  
[178, 145, 104],  
[130, 97, 56],  
[171, 138, 97]], dtype=uint8),
'image/filename': b'004120.jpg',
'labels': array([5], dtype=int64),
'labels_no_difficult': array([5], dtype=int64),
'objects': {'bbox': array([0.12312312, 0.118      , 0.9039039 , 1.        ], dtype=float32),
'is_difficult': array([False]),
'is_truncated': array([False]),
'label': array([5], dtype=int64),
'pose': array([4], dtype=int64)}}
```

典型的一个对象

其中上面是一个图像

labels为5 似乎不是表示box的数量, 而其其中objects就只装了一个box

其中的label和上面的labels似乎是一样的

这里犯了个错误

(?)表示的是一维数组, 也就是label和labels和pose都可以是多个数字, 也即一个图像会有多个标签, 和多个  
bbox

其中objects并非多个对象，其中一个对象就是一组bbox

```
[[ 92, 118, 44],  
 [101, 127, 54],  
 [ 89, 115, 44],  
 ...,  
 [132, 153, 96],  
 [125, 149, 89],  
 [140, 166, 105]],  
  
[[106, 132, 58],  
 [ 95, 121, 48],  
 [ 73,  99, 28],  
 ...,  
 [133, 154, 97],  
 [120, 144, 84],  
 [118, 144, 83]]], dtype=uint8),  
'image/filename': b'007526.jpg',  
'labels': array([12, 14], dtype=int64),  
'labels_no_difficult': array([12, 14], dtype=int64),  
'objects': {'bbox': array([[0.246, 0.50301206, 0.624, 0.78313255],  
 [0.32, 0.25903615, 0.824, 0.99361446]], dtype=float32),  
'is_difficult': array([False, False]),  
'is_truncated': array([False, False]),  
'label': array([14, 12], dtype=int64),  
'pose': array([4, 4], dtype=int64)}
```

这是一个典型的多box

其中labels和label都表示一个idx数组，数字代表类标签，pose似乎是表示形状或比例或姿态？？

后期可能需要把objects对象转换为一个object的数组。

通过把其中的每个变量分离出来

问题：怎么计算yolo的loss，是由数据集决定还是net，loss是否需要分开算，是否需要slice，slice是否被tensorflow支持，包括idx访问，bool arr idx arr访问，是否被tf支持？

tf是否可以在中途改变tensor的shape？

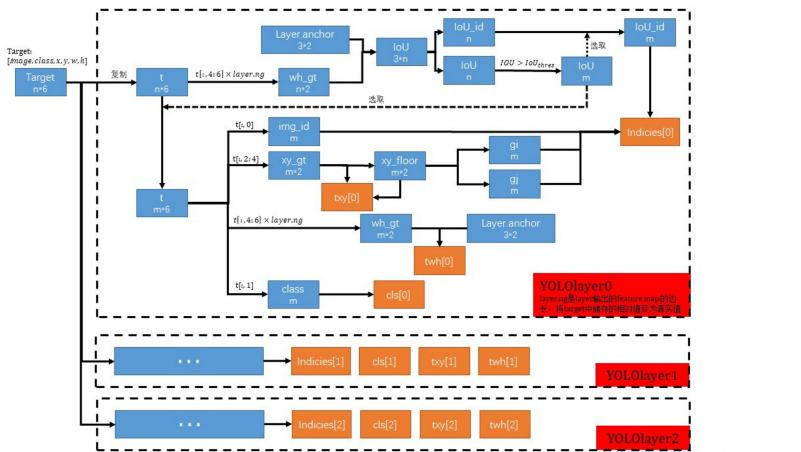
eager execution 是否可以改变tensor的shape？

tensorflow 2 如何同时支持静态和动态图？

研究：关于voc数据集的文件格式，以及自己的数据集（他们使用的格式）的文件格式，以及如何转换为  
voc2007标准格式

关于loss

<https://blog.csdn.net/wqwqqwq1231/article/details/90667046>



可以理解为：loss函数的执行流程？

$$\begin{aligned} loss(object) = & \lambda_{coord} \sum_{i=0}^{K \times K} \sum_{j=0}^M I_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] + \\ & \lambda_{coord} \sum_{i=0}^{K \times K} \sum_{j=0}^M I_{ij}^{obj} (2 - w_i \times h_i) [(w_i - \hat{w}_i)^2 + (h_i - \hat{h}_i)^2] - \\ & \sum_{i=0}^{K \times K} \sum_{j=0}^M I_{ij}^{obj} [\hat{C}_i \log(C_i) + (1 - \hat{C}_i) \log(1 - C_i)] - \\ & \lambda_{noobj} \sum_{i=0}^{K \times K} \sum_{j=0}^M I_{ij}^{noobj} [\hat{C}_i \log(C_i) + (1 - \hat{C}_i) \log(1 - C_i)] - \\ & \sum_{i=0}^{K \times K} I_{ij}^{obj} \sum_{c \in classes} [\hat{p}_i(c) \log(p_i(c)) + (1 - \hat{p}_i(c)) \log(1 - p_i(c))] \end{aligned}$$

loss公式

xhat yhat what hhat 表示正确的box坐标

wi hi 等表示第几个box的坐标

lambda大概表示一个常数

(x-xhat)^2+(y-yhat)^2 表示欧氏距离的平方 等同于欧氏距离的作用

其可以表示为 abs(pred-label)^2 用于n维空间

而l 到 K\*K 和 l 到 M+l

是给x w w h用的，表示box的idx 应该

j是那个l用的 那个表示什么尚不清楚（待调查）

l 的上标表示其是与object对应的，即每个object都有一组l

而每个object有K\*K个w h x y即 K\*K个box

5 第三个维度: [x, y, w, h, class]. x, y 是实际框的中心点坐标, w, h 是框的宽度和高度。x, y, w, h 均是除以图片分辨率得到的[0, 1]范围的值。

而voc数据集中为左上角和右下角坐标

```
[convolutional]
batch_normalize=1
# 是否做 BN 操作
filters=3
# 输出特征图的数量
size=3
# 卷积核的尺寸
stride=1
# 做卷积运算的步长
pad=1
# 如果 pad 为 0, padding 由 padding 参数指定。
# 如果 pad 为 1, padding 大小为 size/2, padding 应该
activation=leaky
```

看起来 3\*3 卷积的激活都是 leaky relu  
并且做 bn

```
[convolutional]
batch_normalize=1
filters=128
size=3
stride=2
pad=1
activation=leaky
```

可以通过带入以上公式, 可以得到 OutFeature 是 InFeature 的一半。  
看说明 此做下采样的 1 1 2 卷积, 可以使用 max pool 替代, 但 1 1 2 卷积可以改变通道, 而 maxpool 不可  
其中 padding 都是使用 "SAME" 进行填充保形的  
这里的下采样卷积也用了 bn  
激活也是 leaky

可以说一个残差块单位里每个层都是 leaky 激活

上采样是通过线性插值实现的。

```
[route]
layers = -1, 36
# 当属性有两个值, 就是将上一层和第 36 层进行 concat
# 即沿深度的维度连接, 这也要求 feature map 大小是一致的。
[route]
layers = -4
# 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。
# 本例子中就是提取从当前倒数第四个层输出
```

这可能是 darknet 中的特殊层? 可以自动选取层输出其输出

```
[shortcut]
from=-3
activation=linear
# shortcut 操作是类似 ResNet 的跨层连接, 参数 from 是 -3,
# 意思是 shortcut 的输出是当前层与先前的倒数第三层相加而得到。
```

可以看到 shortcut 都没有激活, 也就是主体特征抽取全部使用 leaky bn 的卷积, 其他则无激活

这是输出层 最后一个卷积

```
[convolutional]
size=1
stride=1
pad=1
filters=18
# 每一个 [region/yolo] 层前的最后一个卷积层中的
# filters=num(yolo 层个数)*(classes+5) , 5 的意思是 5 个坐标,
# 代表论文中的 tx, ty, tw, th, po
# 这里类别个数为 1, (1+5)*3=18
activation=linear
```

可以看到其是线性输出  
问题: 如何限定范围

```

[yolo]
mask = 6,7,8
# 训练框 mask 的值是 0,1,2,
# 这意味着使用第一, 第二和第三个 anchor
anchors = 10,13, 16,30, 33,23, 30,61, 62,45,
          59,119, 116,90, 156,198, 373,326
# 总共有三个检测层, 共计 9 个 anchor
# 这里的 anchor 是由 kmeans 聚类算法得到的。
classes=1
# 类别个数
num=9
# 每个 grid 预测的 BoundingBox num/yolo 层个数
jitter=.3
# 利用数据抖动产生更多数据,
# 属于 TTA (Test Time Augmentation)
ignore_thresh = .5
# ignore_thresh 指得是参与计算的 IOU 阈值大小。
# 当预测的检测框与 ground true 的 IOU 大于 ignore_thresh 的时候,
# 不会参与 loss 的计算, 否则, 检测框将会参与损失计算。
# 目的是控制参与 loss 计算的检测框的规模, 当 ignore_thresh 过于大,
# 接近于 1 的时候, 那么参与检测框回归 loss 的个数就会比较少, 同时也容易造成过拟合;
# 而如果 ignore_thresh 设置的过于小, 那么参与计算的会数量规模就会很大。
# 同时也容易在进行检测框回归的时候造成欠拟合。
#ignore_thresh 一般选取 0.5-0.7 之间的一个值
# 小尺度 (13*13) 用的是 0.7,
# 大尺度 (26*26) 用的是 0.5。

```

.....

这应该是yolo的特有层

这层中包含了一些似乎是预定义的参数

尚不知道这些参数到底是训练出来的 (可训练

还是预定义好不变的参数

还是说使用某种方法根据每次不同数据来选择的参数

4x	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr><td style="padding: 2px;">Convolutional</td><td style="padding: 2px;">1024</td><td style="padding: 2px;"><math>3 \times 3 / 2</math></td><td style="padding: 2px;"><math>8 \times 8</math></td></tr> <tr><td style="padding: 2px;">Convolutional</td><td style="padding: 2px;">512</td><td style="padding: 2px;"><math>1 \times 1</math></td><td></td></tr> <tr><td style="padding: 2px;">Convolutional</td><td style="padding: 2px;">1024</td><td style="padding: 2px;"><math>3 \times 3</math></td><td></td></tr> <tr><td style="padding: 2px;">Residual</td><td></td><td></td><td style="padding: 2px;"><math>8 \times 8</math></td></tr> </table> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr><td style="padding: 2px;">Avgpool</td><td style="padding: 2px;"></td><td style="padding: 2px;">Global</td></tr> <tr><td style="padding: 2px;">Connected</td><td style="padding: 2px;"></td><td style="padding: 2px;">1000</td></tr> <tr><td style="padding: 2px;">Softmax</td><td style="padding: 2px;"></td><td style="padding: 2px;"></td></tr> </table>	Convolutional	1024	$3 \times 3 / 2$	$8 \times 8$	Convolutional	512	$1 \times 1$		Convolutional	1024	$3 \times 3$		Residual			$8 \times 8$	Avgpool		Global	Connected		1000	Softmax		
Convolutional	1024	$3 \times 3 / 2$	$8 \times 8$																							
Convolutional	512	$1 \times 1$																								
Convolutional	1024	$3 \times 3$																								
Residual			$8 \times 8$																							
Avgpool		Global																								
Connected		1000																								
Softmax																										

[https://blog.csdn.net/DD\\_PP\\_JJ](https://blog.csdn.net/DD_PP_JJ)

darknet53 的结构

最后使用了softmax应该是分类网络

avgpool得到的应该是1024维的向量, 每个通道变成一个标量

connected可能是全连接层? 如果是, 其将1024->1000向量, 并通过softmax输出

1000类的结果

看样子是imagenet上跑的网络

Route层可以方便通过序号和名字获得某一层的输出, tensorflow似乎必须手动保存?

tf无层的概念, 可以通过name获取var, tensor和op, 但不知道是否可以通过指定其名字, 以允许

后续通过名字获取tensor输出?

这样就可以避开路由层, 但这样可能需要改动原有层的代码

在之前的文章中讲过, YOLO 层前一层卷积层的 filter 个数具有特殊的要求, 计算方法为:

$$\text{filter\_num} = \text{anchor\_num} \times (5 + \text{classes\_num})$$

如下图所示:

可能是255通道的来源,  $(5+80) * 3 = 255$  80类 每个img 3个 anchor box

5是 xywhc 其中c为置信度, 此处有东西的置信度

这一层卷积的输出, 由于每个通道的含义都有不同, 需要特殊的激活函数, 分部分激活

, 因此其非普通卷积, 可用卷积+自定义激活函数实现

理解以上内容是需要对应以下公式：

$$b_x = \sigma(t_x) + c_x$$

$$\begin{aligned} b_y &= \sigma(t_y) + c_y \\ b_w &= p_w e^{t_w} \end{aligned}$$

$$b_h = p_h e^{t_h}$$

**xy 部分：**

$$b_x = \sigma(t_x) + c_x$$

$$b_y = \sigma(t_y) + c_y$$

公式，表示了不同部分的及算法方法  
看样子主要还是用sigmoid函数激活

其中b指的是最终输出 c指的是直接输出，没有经过激活计算的

原文如下

理解以上内容是需要对应以下公式：  $b_x = \sigma(t_x) + c_x$   
 $b_y = \sigma(t_y) + c_y$   
 $b_w = p_w e^{t_w}$   
 $b_h = p_h e^{t_h}$   
**xy 部分：**  
 $b_x = \sigma(t_x) + c_x$   
 $b_y = \sigma(t_y) + c_y$   
**c**  
 $x, c$  代表的是格子的左上角坐标；  $t_x, t_y$  代表的是网络预测的结果；  $\sigma$  代表 sigmoid 激活函数。对  
应代码理解：  
`io[..., 2:] = torch.sigmoid(io[..., 2:]) + self.grid_xy`  
**# xy**  
**# grid\_xy** 是左上角再加上偏移量 `io[..., 2:]` 代表 `xy` 偏移  
**wh 部分：**  
GiantPandaCV-prrp  
59从零开始学习 YOLOv3 系列合集  
2020-03-30  
 $b_w = p_w e^{t_w}$   
 $b_h = p_h e^{t_h}$   
**p**  
 $w, ph$  代表的是 anchor 先验框在 feature map 上对应的大小。 $t_w, t_h$  代表的是网络学习得到的缩放  
系数。对应代码理解：  
`# wh yolo method`  
`io[:, 2:4] = torch.exp(io[:, 2:4]) * self.anchor_wh`  
**class 部分：**  
在类别部分，提供了几种方法，根据 arc 参数来进不同的选择。以 CE (crossEntropy) 为例：  
`#io, (bs, anchors, grid, grid, xwh+classes)`  
`io[:, 4:] = F.softmax(io[:, 4:], dim=4)  
io[:, 4:] =`

而重点在于计算 `pw ph`，由于 anchor 框在 label 中是相对于全图的大小，而 feature map 缩小了 n 倍，因此 anchor 框的尺寸也应该缩小 n 倍  
其将很可能会是小数

$t_w t_h$  指宽高缩放系数，这里说明 anchor 先验框只是确定了常用的 box 的尺寸比，而并没有确定大小和位置  
 $t_w t_h$  从网络学习而来，因此应该在 255 通道中存在，但其中似乎并没有  $t_w t_h$  的位置

上面原有错误，应全为  $t_w t_h$

这里澄清， $t_w t_h$  为学习到的参数而非输出，保存在网络中，通过参与 loss 计算而训练

**# wh yolo method**

`io[:, 2:4] = torch.exp(io[:, 2:4]) * self.anchor_wh`

这说明 loss，计算时还需要使用内部参数，因此此处的 loss 函数不再是一个纯函数  
而是一个 layer

上面有个错误

255 通道并非都用 sigmoid 激活

而是在计算 loss 的过程中，根据需要执行激活

其中 class 部分使用 softmax 激活

尺寸部分使用了 exp 函数

xy 部分使用了 sigmoid 函数

可以认为，最后一层没有激活 或使用 linear 激活

而激活和 loss 都在 loss layer 中进行计算

iouLoss =

$$d(box, centroid) = 1 - IoU(box, centroid)$$

或者称作 iou 距离，iou 越大 距离越小

iou 可以看作是重合度

交并比

交集并集 其与重合度很可能不是线性关系

看极端典型，如果两个图像尺寸差距极大，那么当小图在大图中时，iou 并不为 1

而是为小图/大图尺寸，当且仅当两个图的尺寸一样时 iou 才可能为 1，因此

iou 不仅衡量重合度

还大小相似的，可以认为是同时考虑了尺寸和位置的匹配度

但其是否公平地考虑了尺寸和位置

其是否有对尺寸和位置都有二次方特性？即距离越远，误差值会非线性急剧上升？

这需要考虑，位置的接近和尺寸的接近对 iou 的贡献是否是一样的

即位置从 x 到 x+h 时的贡献（远离或接近），与 w h 的接近和远离时的 iou 值的变化情况，即可

假设  $x+1, y+1$  是  $x, y$  都更接近，那么其并集不变，交集变大，基本为一行+一列的像素（重合部分）设重合部分为  $ih$  和  $iw$ ，那么  $ih$  对应  $y$ ， $iw$  对应  $x$ ，纵向增加为  $ih$

横向增加为  $iw$ ，还要再加上角落的一个像素，为  $ih+iw+1$ ，可暂时忽略不计，为  $ih+iw$

那么当从  $x, y$  变化到  $x+t_x, y+t_y$  时，就是一个差分过程， $ih+iw+ih+1+...+ih+n-1+iw+n-1$  那么 额外有  $2*(1+2+...+n-1)$  再加上  $(ih+iw)*n$

而当某个图片的 wh 变化得最近时虽然并不一定会有什么变化，其只改变了分子，而分母一般是比较大的，当分子即重合度比较小时，改变分母的结果影响很小

当不重合时则完全没有影响，可以说

有待研究

