Yolov3&Yolov4核心基礎知識完整講解

小白 小白學視覺 2022-03-29 10:05

作者：**江大白**

知乎鏈接：**https://zhuanlan.zhihu.com/p/143747206**

文章目錄

1.論文匯總

- Yolov3論文名：《Yolov3: An Incremental Improvement》
- Yolov3論文地址：https://arxiv.org/pdf/1804.02767.pdf
- Yolov4論文名：《Yolov4: Optimal Speed and Accuracy of Object Detection》
- Yolov4論文地址：https://arxiv.org/pdf/2004.10934.pdf

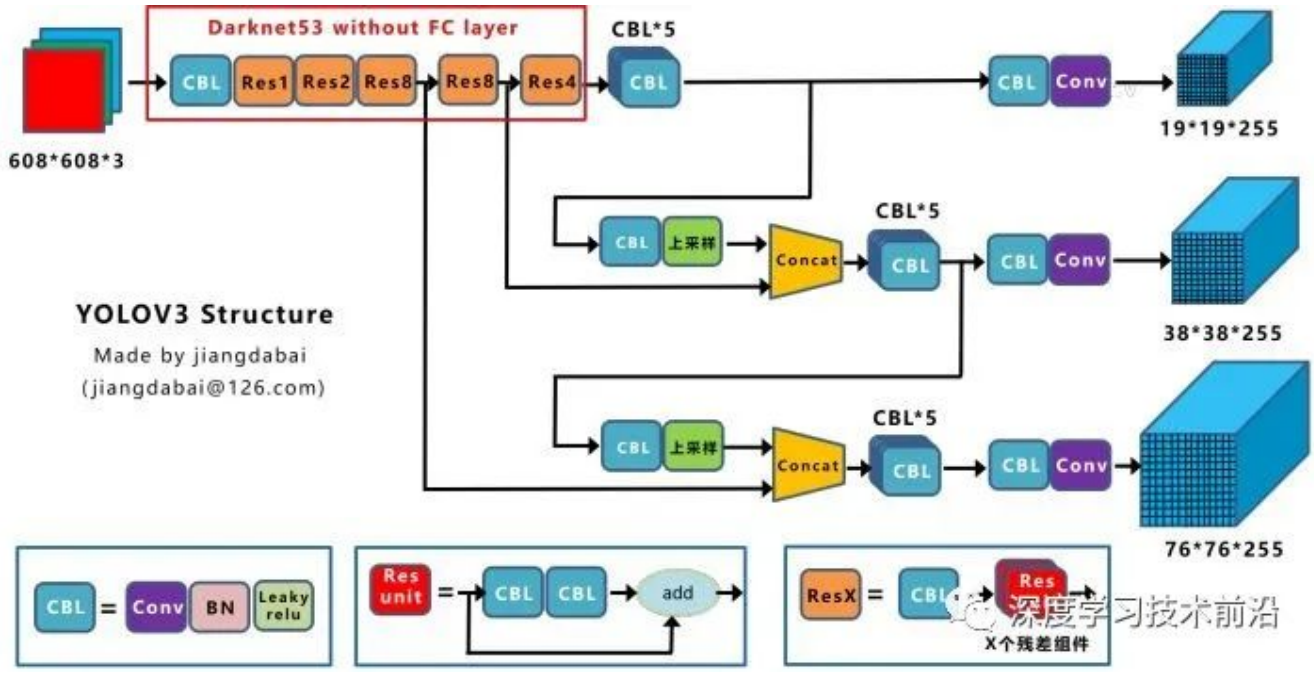
2.YoloV3核心基礎內容

2.1 網絡結構可視化

Yolov3是目標檢測**Yolo系列**非常非常經典的算法，不過很多同學拿到**Yolov3**或者**Yolov4**的**cfg文件**時，並不知道如何直觀的可視化查看網絡結構。如果純粹看cfg裡面的內容，肯定會**一臉懵逼**。

其實可以很方便的用**netron**查看**Yolov3的網絡結構圖**，一目了然。

2.2 網絡結構圖



上圖三個藍色方框內表示Yolov3的三個基本組件：

CBL： Yolov3網絡結構中的最小組件，由**Conv+Bn+Leaky_relu**激活函數三者組成。

Res unit：借鑒**Resnet**網絡中的殘差結構，讓網絡可以構建的更深。
ResX：由一個**CBL**和**X**個殘差組件構成，是Yolov3中的大組件。每個Res模塊前面的CBL都起到下採樣的作用，因此經過5次Res模塊後，得到的特徵圖是**608->304->152->76->38->19大小**。

其他基礎操作：

Concat：張量拼接，會擴充兩個張量的維度，例如26*26*256和26*26*512兩個張量拼接，結果是26*26*768。Concat和cfg文件中的route功能一樣。
add：張量相加，張量直接相加，不會擴充維度，例如104*104*128和104*104*128相加，結果還是104*104*128。add和cfg文件中的shortcut功能一樣。

Backbone中卷積層的數量：

每個ResX中包含1+2*X個卷積層，因此整個主幹網絡Backbone中一共包含
1+（1+2*1）+（1+2*2）+（1+2*8）+（1+2*8）+（1+2*4）=52，再加上一個FC全連接層，即可以組成一個**Darknet53分類網絡。**不過在目標檢測Yolov3中，去掉FC層，不過為了方便稱呼，仍然把**Yolov3**的主幹網絡叫做**Darknet53結構。**

2.3 核心基礎內容

Yolov3是2018年發明提出的，這成為了目標檢測**one-stage**中非常經典的算法，包含**Darknet-53網絡結構、anchor錨框、FPN**等非常優秀的結構。

本文主要目的在於描述**Yolov4**和**Yolov3**算法的不同及創新之處，對Yolov3的基礎不過多描述。

不過大白也正在準備Yolov3算法非常淺顯易懂的基礎視頻課程，讓小白也能簡單清楚的了解Yolov3的整個過程及各個算法細節，製作好後會更新到此處，便於大家查看。

在準備課程過程中，大白蒐集查看了網絡上幾乎所有的Yolov3資料，在此整理幾個非常不錯的文章及視頻，大家也可以點擊查看，學習相關知識。

（1）視頻：吳恩達目標檢測Yolo入門講解

- https://www.bilibili.com/video/BV1N4411J7Y6?from=search&seid=18074481568368507115

（2）文章：Yolo系列之Yolov3【深度解析】

- https://blog.csdn.net/leviopku/article/details/82660381

（3）文章：一文看懂Yolov3

- https://blog.csdn.net/litt1e/article/details/88907542

相信大家看完，對於**Yolov3的基礎知識點**會有一定的了解。

3.YoloV3相關代碼

3.1 python代碼

代碼地址：https://github.com/ultralytics/Yolov3

3.2 C++代碼

這裡推薦Yolov4作者的darknetAB代碼，代碼和原始作者代碼相比，進行了很多的優化，如需要運行Yolov3網絡，加載cfg時，使用Yolov3.cfg即可

代碼地址：https://github.com/AlexeyAB/darknet

3.3 python版本的Tensorrt代碼

除了算法研究外，實際項目中還需要將算法落地部署到工程上使用，比如GPU服務器使用時還需要對模型進行tensorrt加速。

（1）Tensort中的加速案例

強烈推薦tensort軟件中，自帶的Yolov3加速案例，路徑位於tensorrt解壓文件夾的TensortX/samples/python/Yolov3_onnx中

針對案例中的代碼，如果有不明白的，也可參照下方文章上的詳細說明：

代碼地址：https://www.cnblogs.com/shouhuxianjian/p/10550262.html

（2）Github上的tensorrt加速

除了tensorrt軟件中的代碼， github上也有其他作者的開源代碼

代碼地址：https://github.com/lewes6369/TensorRT-Yolov3

3.4 C++版本的Tensorrt代碼

項目的工程部署上，如果使用C++版本進行Tensorrt加速，一方面可以參照Alexey的github代碼，另一方面也可以參照下面其他作者的開源代碼

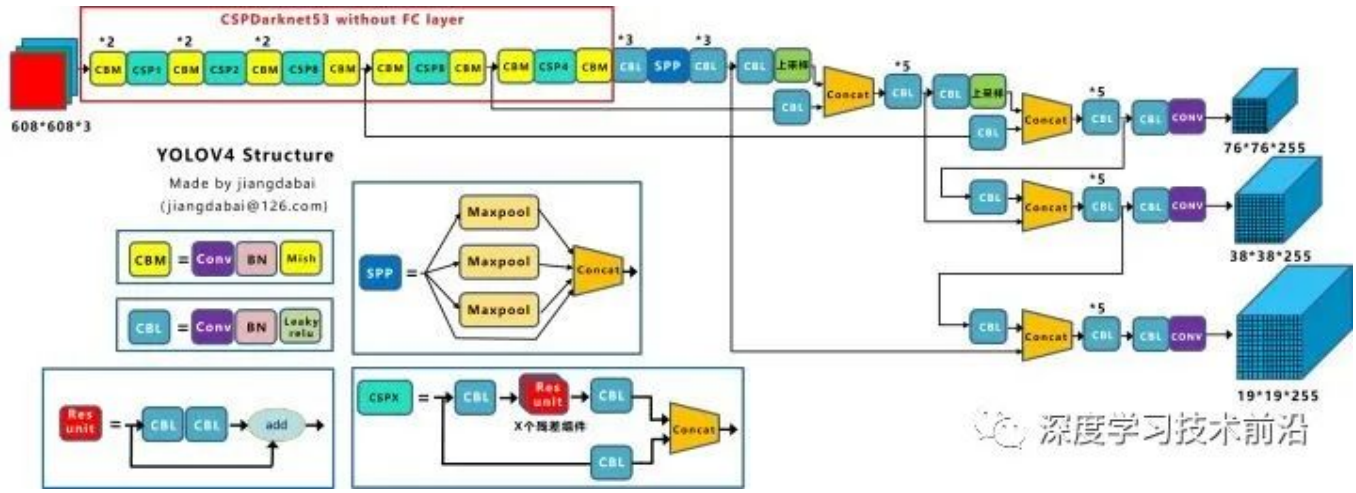
代碼地址：https://github.com/wang-xinyu/tensorrtx/tree/master/Yolov3

4.YoloV4核心基礎內容

4.1 網絡結構可視化

Yolov4的網絡結構也可以使用netron工具查看。

4.2 網絡結構圖



Yolov4的結構圖和Yolov3相比，因為多了**CSP結構，PAN結構**，如果單純看可視化流程圖，會覺得很繞，不過在繪製出上面的圖形後，會覺得豁然開朗，其實整體架構和Yolov3是相同的，不過使用各種新的算法思想對各個子結構都進行了改進。

先整理下Yolov4的五個基本組件：

CBM： Yolov4網絡結構中的最小組件，由Conv+Bn+Mish激活函數三者組成。
CBL：由Conv+Bn+Leaky_relu激活函數三者組成。
Res unit：借鑒Resnet網絡中的殘差結構，讓網絡可以構建的更深。
CSPX：借鑒CSPNet網絡結構，由三個卷積層和X個Res unint模塊Concate組成。
SPP：採用1×1，5×5，9×9，13×13的最大池化的方式，進行多尺度融合。

其他基礎操作：

Concat：張量拼接，維度會擴充，和Yolov3中的解釋一樣，對應於cfg文件中的route操作。
add：張量相加，不會擴充維度，對應於cfg文件中的shortcut操作。

Backbone中卷積層的數量：

和Yolov3一樣，再來數一下Backbone裡面的捲積層數量。

每個CSPX中包含3+2*X個卷積層，因此整個主幹網絡Backbone中一共包含
2+（3+2*1）+2+（3+2*2）+2+（3+2*8）+2+（3+2*8）+2+（3+2*4）+1=72。

這里大白有些疑惑，按照Yolov3設計的傳統，這麼多卷積層，主幹網絡不應該叫**CSPDaeknet73嗎？？？？**

4.3 核心基礎內容

Yolov4本質上和**Yolov3**相差不大，可能有些人會覺得失望。

但我覺得算法創新分為三種方式：

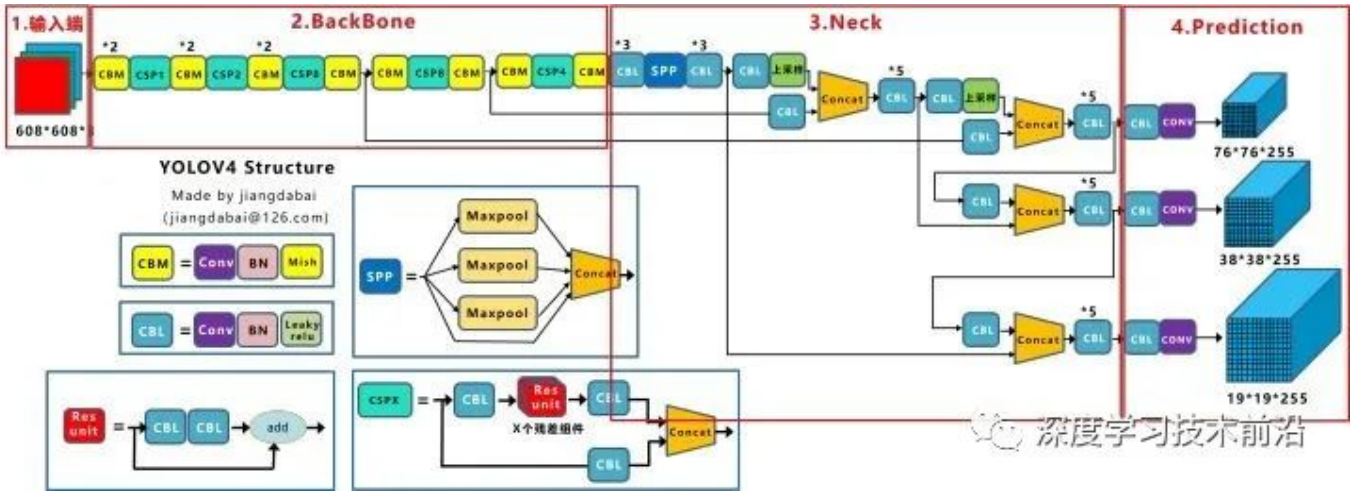
第一種：面目一新的創新，比如Yolov1、Faster-RCNN、Centernet等，開創出新的算法領域，不過這種也是最難的
第二種：守正出奇的創新，比如將圖像金字塔改進為特徵金字塔
第三種：各種先進算法集成的創新，比如不同領域發表的最新論文的tricks，集成到自己的算法中，卻發現有出乎意料的改進

Yolov4既有第二種也有第三種創新，組合嘗試了大量深度學習領域最新論文的20多項研究成果，而且不得不佩服的是作者**Alexey**在**github**代碼庫維護的頻繁程度。

目前Yolov4代碼的star數量已經**1萬多**，據我所了解，目前超過這個數量的，目標檢測領域只有**Facebook的Detectron(v1-v2)**、和**Yolo(v1-v3)官方代碼庫（已停止更新）。**

所以**Yolov4**中的各種創新方式，大白覺得還是很值得仔細研究的。

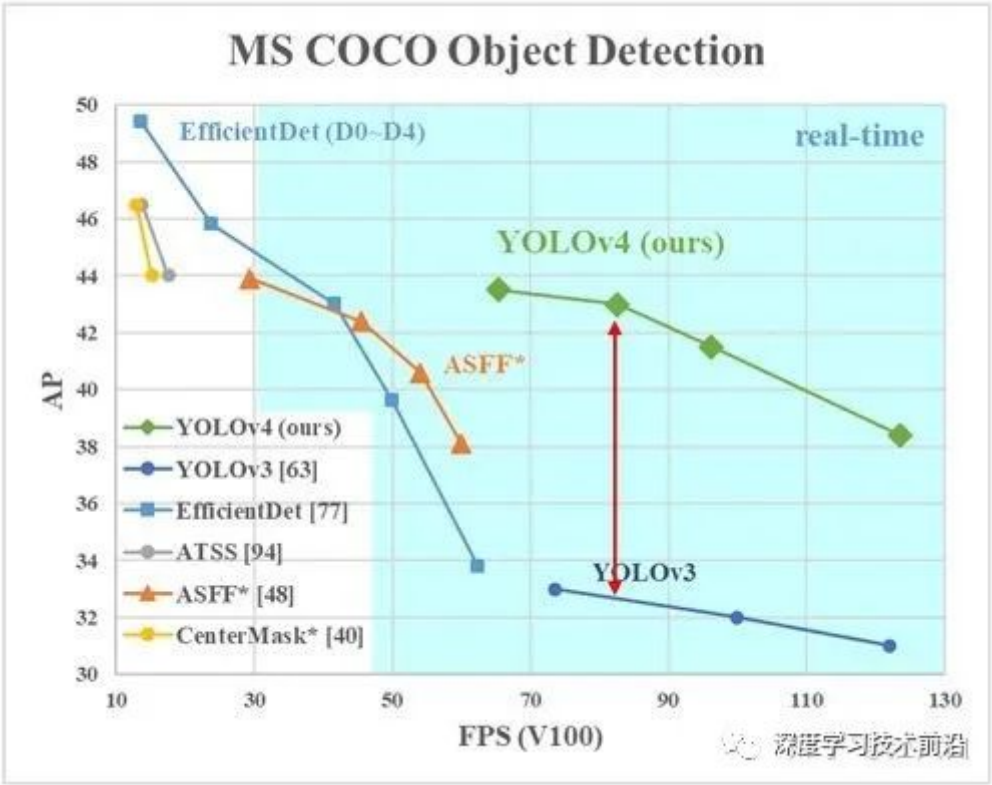
為了便於分析，將Yolov4的整體結構拆分成四大板塊：



大白主要從以上4個部分對YoloV4的創新之處進行講解，讓大家一目了然。

輸入端：這裡指的創新主要是訓練時對輸入端的改進，主要包括**Mosaic數據增強、cmBN、SAT自對抗訓練**
BackBone主幹網絡：將各種新的方式結合起來，包括：**CSPDarknet53、Mish激活函數、Dropblock**
Neck：目標檢測網絡在BackBone和最後的輸出層之間往往會插入一些層，比如Yolov4中的**SPP模塊、FPN+PAN結構**
Prediction：輸出層的錨框機制和Yolov3相同，主要改進的是訓練時的損失函數**CIOU_Loss**，以及預測框篩選的nms變為**DIOU_nms**

總體來說，**Yolov4對Yolov3的各個部分都進行了改進優化**，下面丟上作者的算法對比圖。



僅對比**Yolov3和Yolov4**，在COCO數據集上，同樣的FPS等於83左右時，Yolov4的AP是43，而Yolov3是33，直接上漲了**10個百分點。**

不得不服，當然可能針對具體不同的數據集效果也不一樣，但總體來說，改進效果是很優秀的，下面大白對**Yolov4的各個創新點**繼續進行深挖。

4.3.1 輸入端創新

考慮到很多同學GPU顯卡數量並不是很多，**Yolov4**對訓練時的輸入端進行改進，使得訓練在單張GPU上也能有不錯的成績。比如**數據增強Mosaic、cmBN、SAT自對抗訓練。**

但感覺cmBN和SAT影響並不是很大，所以這裡主要講解Mosaic數據增強。

（1）Mosaic數據增強

Yolov4中使用的**Mosaic**是參考2019年底提出的**CutMix數據增強**的方式，但**CutMix**只使用了兩張圖片進行拼接，而**Mosaic數據增強**則採用了4張圖片，**隨機縮放、隨機裁剪、隨機排布**的方式進行拼接。

這里首先要了解為什麼要進行**Mosaic數據增強**呢？

在平時項目訓練時，**小目標的AP**一般比中目標和大目標低很多。而Coco數據集中也包含大量的小目標，但比較麻煩的是小目標的分佈**並不均勻**。

首先看下小、中、大目標的定義：
2019年發布的論文《Augmentation for small object detection》對此進行了區分：

	Min rectangle area	Max rectangle area
Small object	0*0	32*32
Medium object	32*32	96*96
Large object	96*96	

可以看到小目標的定義是目標框的長寬**0×0~32×32**之間的物體。

	Small	Mid	Large
Ratio of total boxes(%)	41.4	34.3	24.3
Ratio of images included(%)	52.3	70.	3.

但在整體的數據集中，小、中、大目標的佔比並不均衡。
如上表所示，Coco數據集中小目標佔比達到**41.4%**，數量比中目標和大目標都要多。

但在所有的訓練集圖片中，只有**52.3%**的圖片有小目標，而中目標和大目標的分佈相對來說更加均勻一些。

針對這種狀況，Yolov4的作者採用了**Mosaic數據增強**的方式。

主要有幾個優點：

> **豐富數據集：**隨機使用**4張圖片**，隨機縮放，再隨機分佈進行拼接，大大豐富了檢測數據集，特別是隨機縮放增加了很多小目標，讓網絡的魯棒性更好。
> **減少GPU：**可能會有人說，隨機縮放，普通的數據增強也可以做，但作者考慮到很多人可能只有一個GPU，因此Mosaic增強訓練時，可以直接計算4張圖片的數據，使得Mini-batch大小並不需要很大，一個GPU就可以達到比較好的效果。

此外，發現**另一研究者的訓練方式**也值得借鑒，採用的數據增強和Mosaic比較類似，也是使用**4張圖片**（**不是隨機分佈**），但訓練計算loss時，採用“**缺啥補啥**”的思路：

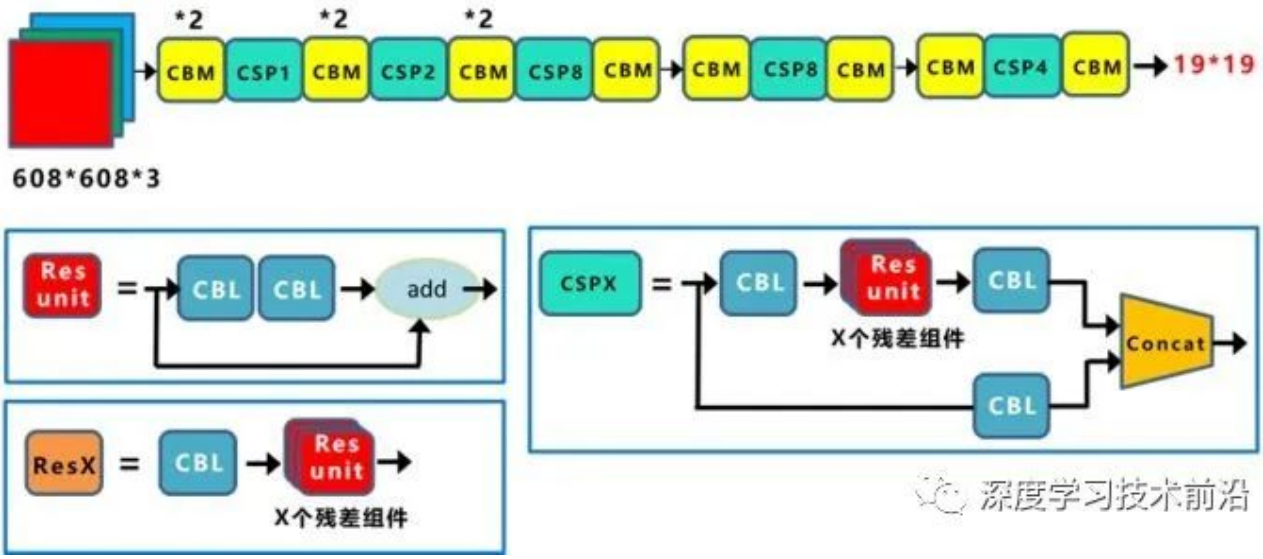
如果上一個iteration中，小物體產生的**loss不足**（比如小於某一個閾值），則下一個iteration就用**拼接圖**；否則就用**正常圖片**訓練，也很有意思。

參考鏈接：https://www.zhihu.com/question/390191723?rf=390194081

4.3.2 BackBone創新

（1）CSPDarknet53

CSPDarknet53是在Yolov3主幹網絡**Darknet53**的基礎上，借鑒**2019年CSPNet**的經驗，產生的**Backbone**結構，其中包含了**5個CSP**模塊。



這裡因為**CSP模塊**比較長，不放到本處，大家也可以點擊Yolov4的netron網絡結構圖，對比查看，一目了然。

每個CSP模塊前面的捲積核的大小都是3*3，stride=2，因此可以起到下採樣的作用。

因為Backbone有5個**CSP模塊**，輸入圖像是**608*608**，所以特徵圖變化的規律是：**608->304->152->76->38->19**

經過5次CSP模塊後得到19*19大小的特徵圖。

而且作者只在Backbone中採用了**Mish激活函數**，網絡後面仍然採用**Leaky_relu激活函數。**

我們再看看下作者為啥要參考2019年的**CSPNet**，採用CSP模塊?

CSPNet論文地址：https://arxiv.org/pdf/1911.11929.pdf

CSPNet全稱是Cross Stage Paritial Network，主要從網絡結構設計的角度解決推理中從計算量很大的問題。

CSPNet的作者認為推理計算過高的問題是由於網絡優化中的**梯度信息重複**導致的。

因此採用CSP模塊先將基礎層的特徵映射劃分為兩部分，然後通過跨階段層次結構將它們合併，在減少了計算量的同時可以保證準確率。

因此Yolov4在主幹網絡Backbone採用CSPDarknet53網絡結構，主要有三個方面的優點：

優點一：增強CNN的學習能力，使得在輕量化的同時保持準確性。

優點二：降低計算瓶頸

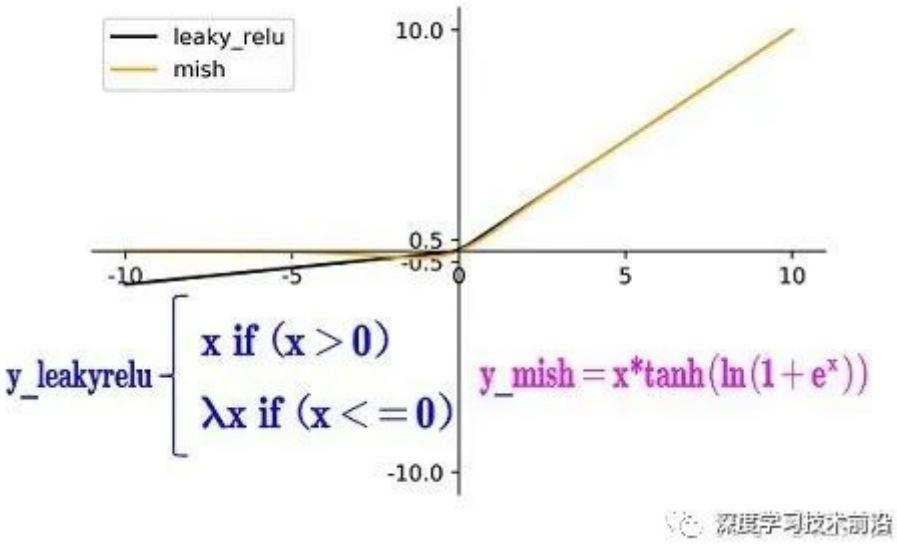
優點三：降低內存成本

（2）Mish激活函數

Mish激活函數是**2019年下半年**提出的激活函數

論文地址：https://arxiv.org/abs/1908.08681

和**Leaky_relu激活函數**的圖形對比如下：



Yolov4的**Backbone**中都使用了**Mish激活函數**，而後面的網絡則還是使用leaky_relu函數。

MixUp	CutMix	Mosaic	Bluring	Label Smoothing	Swish	Mish	Top-1	Top-5
							77.2%	93.6%
✓	✓			✓			**77.8%**	**94.4%**
✓	✓			✓				

Yolov4作者實驗測試時，使用**CSPDarknet53**網絡在**ImageNet數據集**上做圖像分類任務，發現使用了Mish激活函數的**TOP-1**和**TOP-5**的精度比沒有使用時都略高一些。

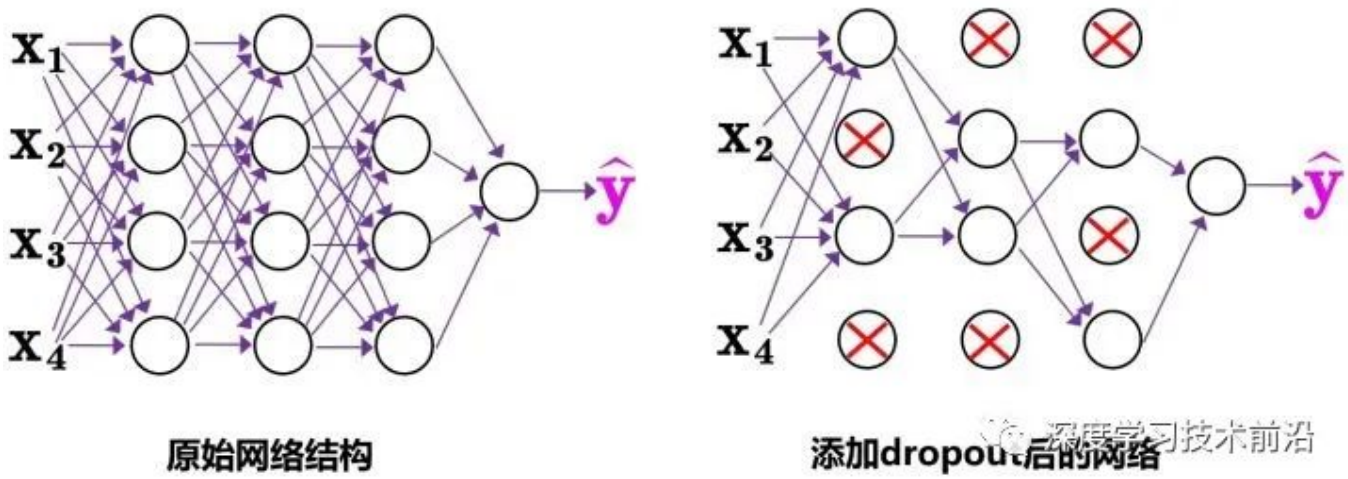
因此在設計Yolov4目標檢測任務時，主幹網絡Backbone還是使用**Mish激活函數。**

（3）Dropblock

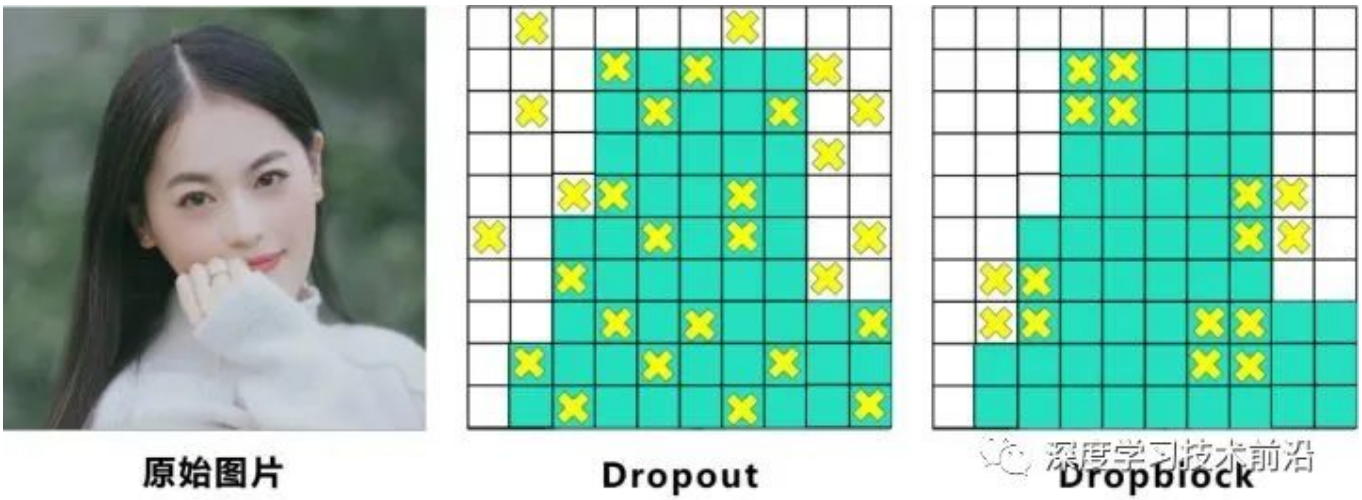
Yolov4中使用的**Dropblock**，其實和常見網絡中的Dropout功能類似，也是緩解過擬合的一種正則化方式。

Dropblock在2018年提出，論文地址：https://arxiv.org/pdf/1810.12890.pdf

傳統的Dropout很簡單，一句話就可以說的清：**隨機刪除減少神經元的數量，使網絡變得更簡單。**



而Dropblock和Dropout相似，比如下圖：



中間Dropout的方式會隨機的刪減丟棄一些信息，但**Dropblock的研究者**認為，卷積層對於這種隨機丟棄並不敏感，因為卷積層通常是三層連用：**卷積+激活+池化層**，池化層本身就是對相鄰單元起作用。而且即使隨機丟棄，卷積層仍然可以從相鄰的激活單元學習到**相同的信息**。

因此，在全連接層上效果很好的Dropout在卷積層上**效果並不好**。

所以**右圖Dropblock的研究者**則乾脆整個局部區域進行刪減丟棄。

這種方式其實是藉鑑**2017年的cutout數據增強**的方式，cutout是將輸入圖像的部分區域清零，而Dropblock則是將Cutout應用到每一個特徵圖。而且並不是用固定的歸零比率，而是在訓練時以一個小的比率開始，隨著訓練過程**線性的增加這個比率**。

CIFAR-10 上Cutout数据增 深度学习技术前沿

Dropblock的研究者與**Cutout**進行對比驗證時，發現有幾個特點：

優點一： Dropblock的效果優於Cutout

優點二： Cutout只能作用於輸入層，而Dropblock則是將Cutout應用到網絡中的每一個特徵圖上

優點三： Dropblock可以定制各種組合，在訓練的不同階段可以修改刪減的概率，從空間層面和時間層面，和Cutout相比都有更精細的改進。

Yolov4中直接採用了更優的**Dropblock**，對網絡的正則化過程進行了全面的升級改進。

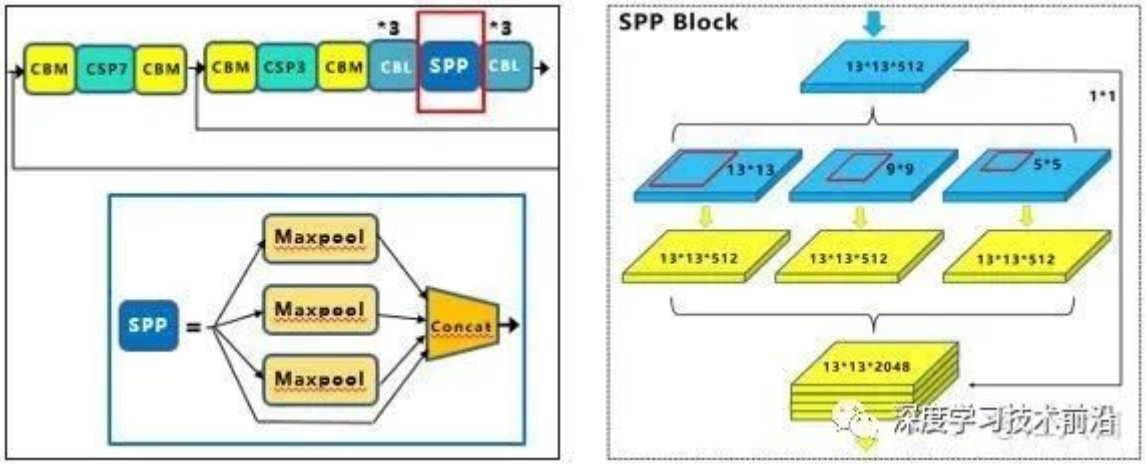
4.3.3 Neck創新

在目標檢測領域，為了更好的提取融合特徵，通常在**Backbone**和**輸出層**，會插入一些層，這個部分稱為Neck。相當於目標檢測網絡的頸部，也是非常關鍵的。

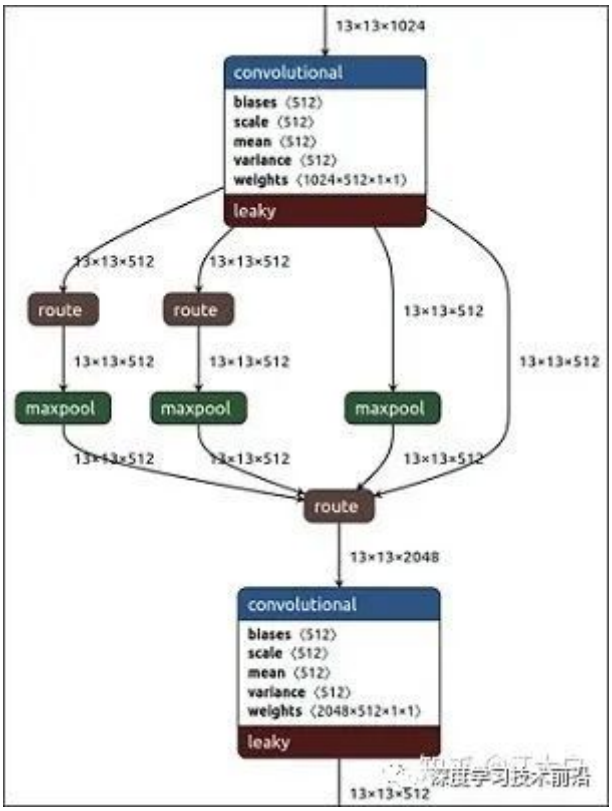
Yolov4的Neck結構主要採用了**SPP模塊**、**FPN+PAN**的方式。

（1）SPP模塊

SPP模塊，其實在Yolov3中已經存在了，在**Yolov4**的C++代碼文件夾**中**有一個**Yolov3_spp版本**，但有的同學估計從來沒有使用過，在Yolov4中，SPP模塊仍然是在Backbone主幹網絡之後：



作者在SPP模塊中，使用k={1*1,5*5,9*9,13*13}的最大池化的方式，再將不同尺度的特徵圖進行Concat操作。



在2019提出的《**DC-SPP-Yolo**》文章：
https://arxiv.org/ftp/arxiv/papers/1903/1903.08589.pdf

也對Yolo目標檢測的**SPP模塊**進行了對比測試。

和Yolov4作者的研究相同，採用**SPP模塊**的方式，比單純的使用**k*k最大池化**的方式，更有效的增加主幹特徵的接收範圍，顯著的分離了最重要的上下文特徵。

Yolov4的作者在使用**608*608**大小的圖像進行測試時發現，在COCO目標檢測任務中，以0.5%的額外計算代價將AP50增加了2.7%，因此Yolov4中也採用了**SPP模塊。**

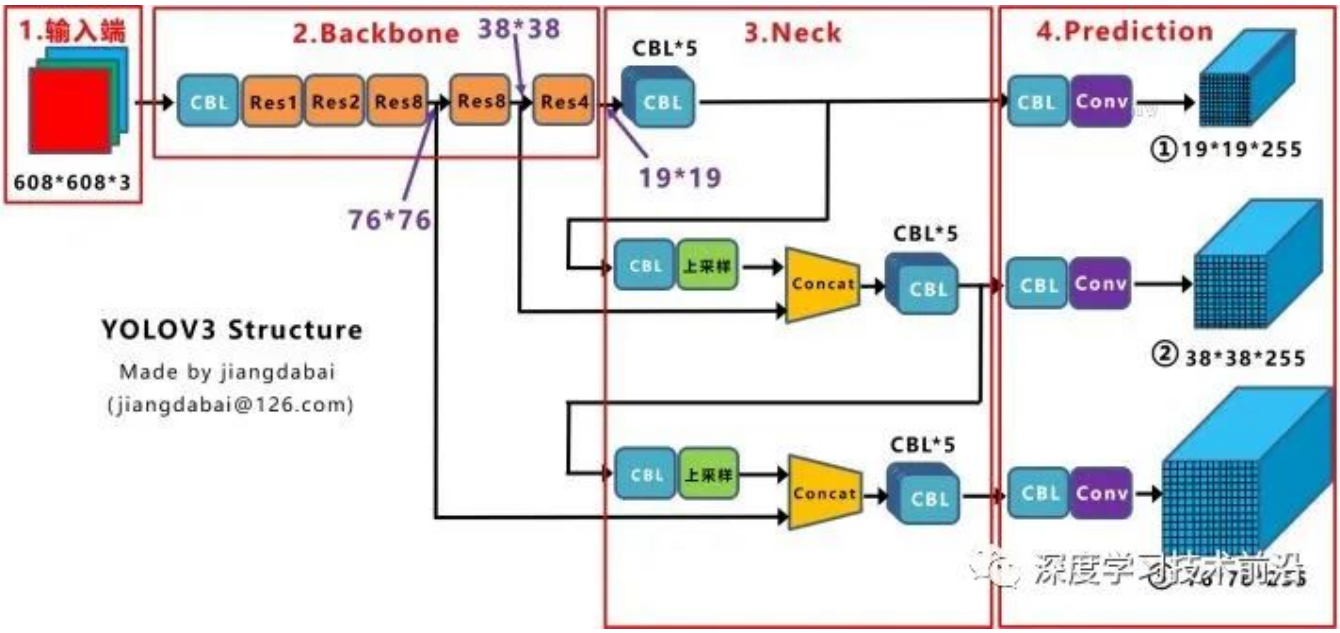
（2）FPN+PAN

PAN結構比較有意思，看了網上Yolov4關於這個部分的講解，大多都是講的比較籠統的，而PAN是藉鑑圖像分割領域PANet的創新點，有些同學可能不是很清楚。

下面大白將這個部分拆解開來，看下Yolov4中是如何設計的。

Yolov3結構：

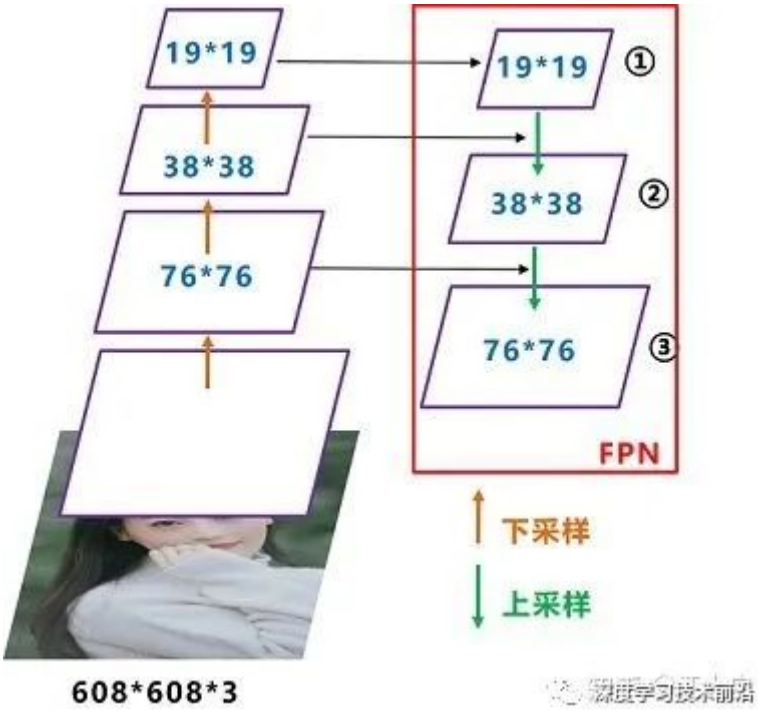
我們先來看下Yolov3中Neck的FPN結構



可以看到經過幾次下採樣，三個紫色箭頭指向的地方，輸出分別是**76*76、38*38、19*19。**

以及最後的**Prediction**中用於預測的三個特徵圖**①19*19*255、②38*38*255、③76*76*255。**
[注：255表示80類別(1+4+80)×3=255]

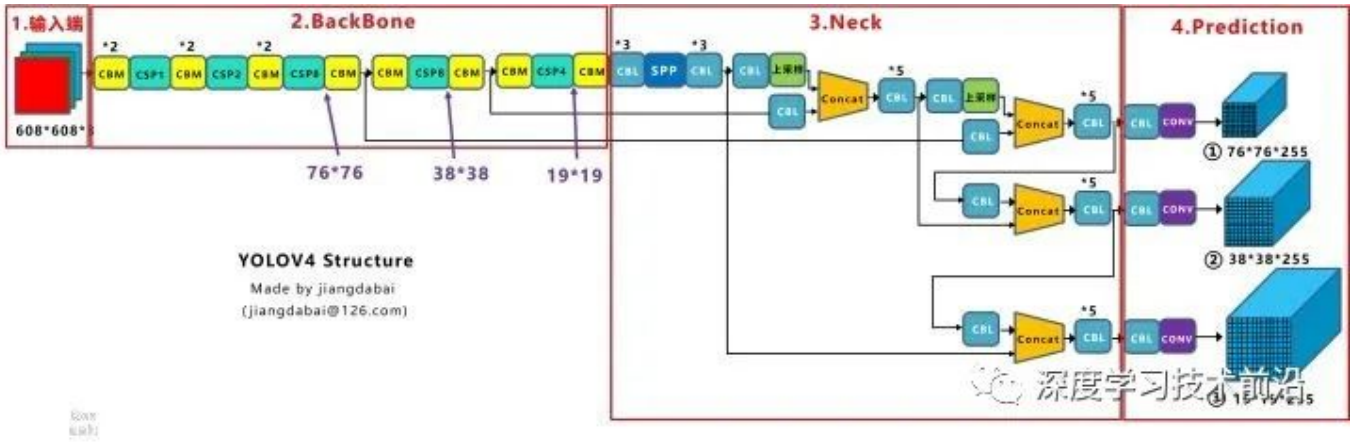
我們將Neck部分用立體圖畫出來，更直觀的看下兩部分之間是如何通過**FPN結構**融合的。



如圖所示，FPN是自頂向下的，將高層的特徵信息通過**上採樣**的方式進行傳遞融合，得到進行預測的特徵圖。

Yolov4結構：

而Yolov4中Neck這部分除了使用FPN外，還在此基礎上使用了PAN結構：

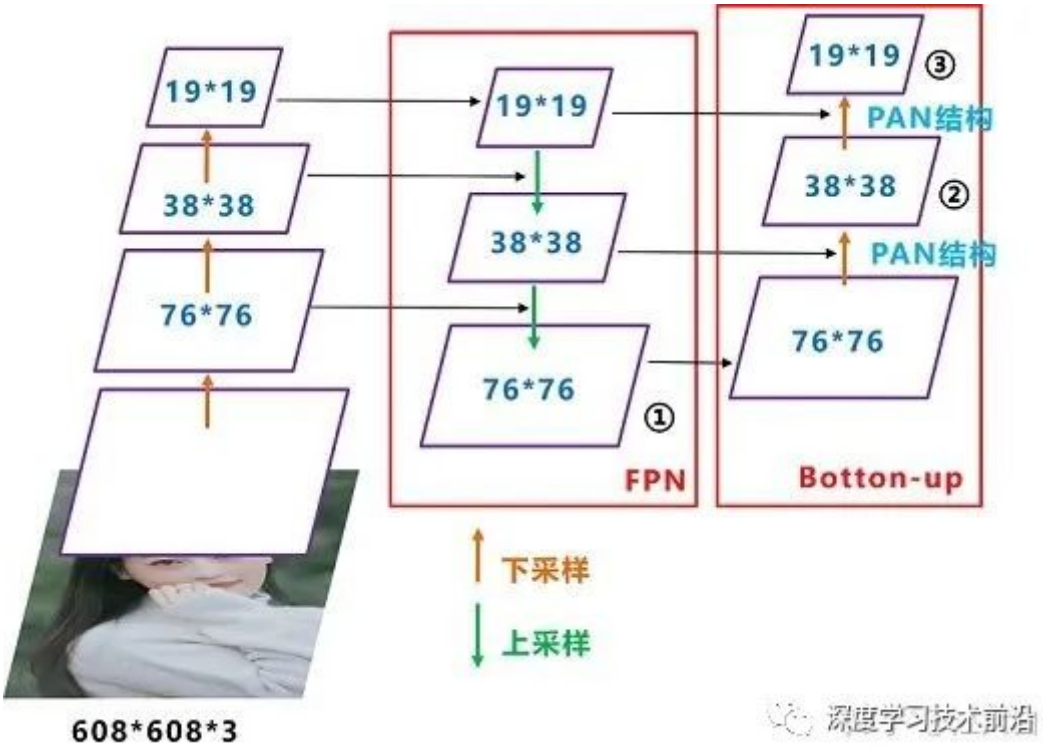


前面CSPDarknet53中講到，每個CSP模塊前面的捲積核都是**3*3大小**，相當於下採樣操作。

因此可以看到三個紫色箭頭處的特徵圖是**76*76、38*38、19*19** 。

以及最後Prediction中用於預測的三個特徵圖：**①76*76*255，②38*38*255，③19*19*255。**

我們也看下**Neck**部分的立體圖像，看下兩部分是如何通過**FPN+PAN結構**進行融合的。



和Yolov3的FPN層不同，Yolov4在FPN層的後面還添加了一個**自底向上的特徵金字塔。**

其中包含兩個**PAN結構。**

這樣結合操作，FPN層自頂向下傳達**強語義特徵**，而特徵金字塔則自底向上傳達**強定位特徵**，兩兩聯手，從不同的主幹層對不同的檢測層進行參數聚合,這樣的操作確實很皮。

FPN+PAN借鑒的是18年CVPR的**PANet**，當時主要應用於**圖像分割領域**，但Alexey將其拆分應用到Yolov4中，進一步提高特徵提取的能力。

不過這裡需要注意幾點：

注意一：

Yolov3的FPN層輸出的三個大小不一的特徵圖①②③直接進行預測

但Yolov4的FPN層，只使用最後的一個76*76特徵圖①，而經過兩次PAN結構，輸出預測的特徵圖②和③。

這裡的不同也體現在cfg文件中，這一點有很多同學之前不太明白，

比如Yolov3.cfg最後的三個Yolo層，

第一個Yolo層是最小的特徵圖**19*19**，mask= **6,7,8**，對應**最大的anchor box**。

第二個Yolo層是中等的特徵圖**38*38**，mask= **3,4,5**，對應**中等的anchor box**。

第三個Yolo層是最大的特徵圖**76*76**，mask= **0,1,2**，對應**最小的anchor box**。

而Yolov4.cfg則**恰恰相反**

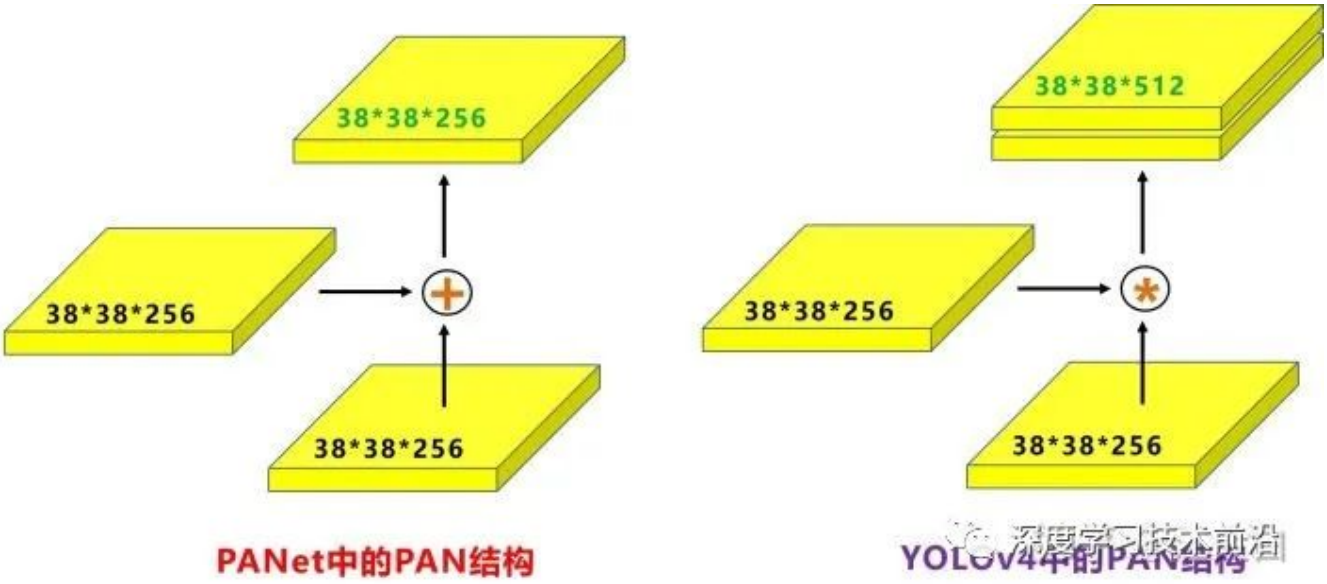
第一個Yolo層是最大的特徵圖**76*76**，mask= **0,1,2**，對應**最小的anchor box**。

第二個Yolo層是中等的特徵圖**38*38**，mask= **3,4,5**，對應**中等的anchor box**。

第三個Yolo層是最小的特徵圖**19*19**，mask= **6,7,8**，對應**最大的anchor box**。

注意點二：

原本的PANet網絡的**PAN結構**中，兩個特徵圖結合是採用**shortcut**操作，而Yolov4中則採用**concat（route）**操作，特徵圖融合後的尺寸發生了變化。



這裡也可以對應Yolov4的netron網絡圖查看，很有意思。

4.3.4 Prediction創新

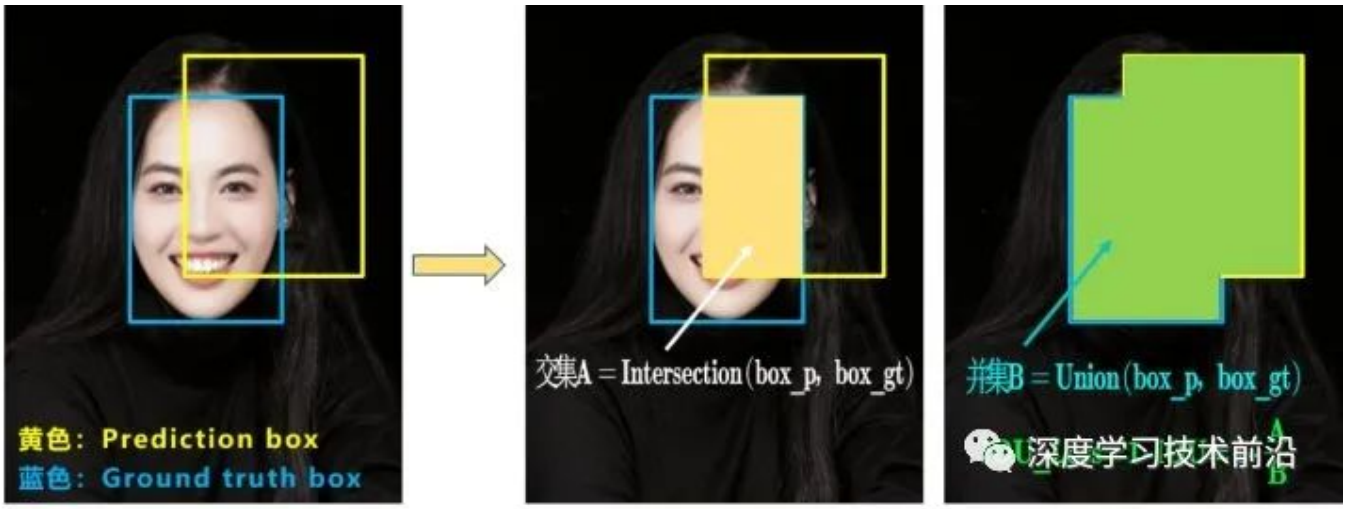
（1）CIOU_loss

目標檢測任務的損失函數一般由**Classificition Loss（分類損失函數）**和**Bounding Box Regeression Loss（回歸損失函數）**兩部分構成。

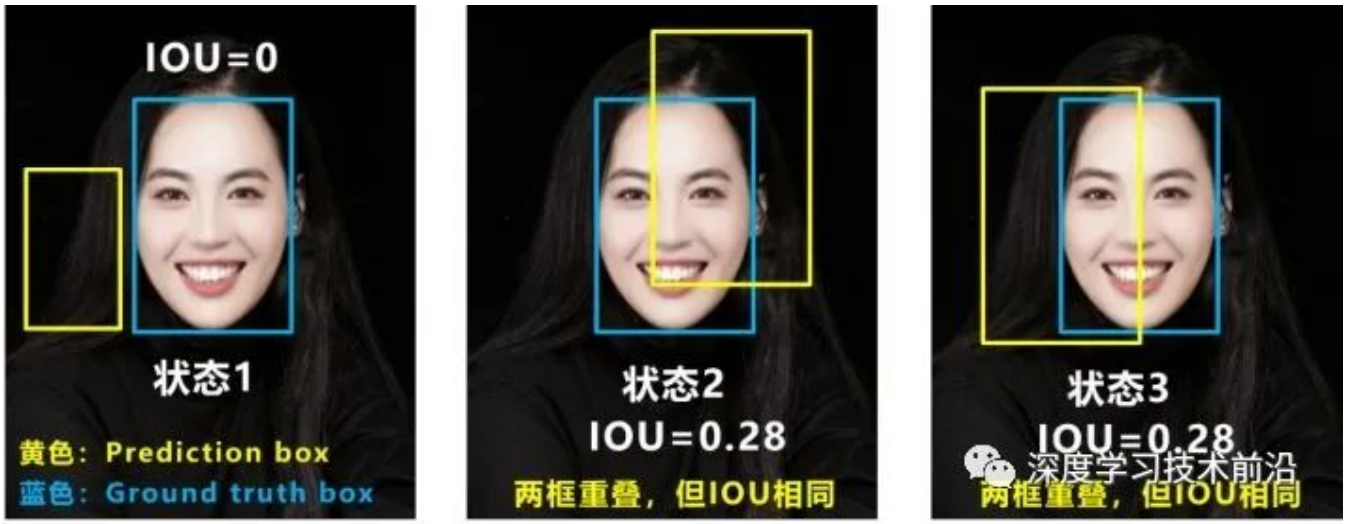
Bounding Box Regeression的Loss近些年的發展過程是：**Smooth L1 Loss-> IoU Loss（2016）-> GIoU Loss（2019）-> DIoU Loss（2020）->CIoU Loss（2020）**

我們從最常用的**IOU_Loss**開始，進行對比拆解分析，看下Yolov4為啥要選擇**CIOU_Loss**。

a.IOU_Loss



可以看到IOU的loss其實很簡單，主要是**交集/並集**，但其實也存在兩個問題。

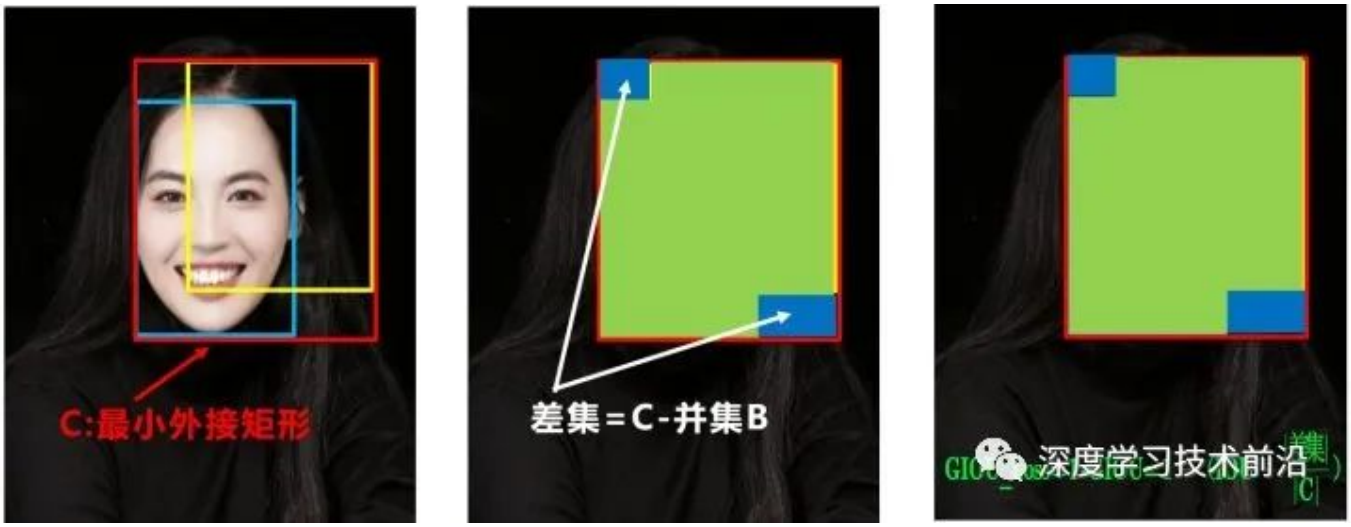


問題1：即狀態1的情況，當預測框和目標框不相交時，IOU=0，無法反應兩個框距離的遠近，此時損失函數不可導，IOU_Loss無法優化兩個框不相交的情況。

問題2：即狀態2和狀態3的情況，當兩個預測框大小相同，兩個IOU也相同，IOU_Loss無法區分兩者相交情況的不同。

因此**2019**年出現了GIOU_Loss來進行改進。

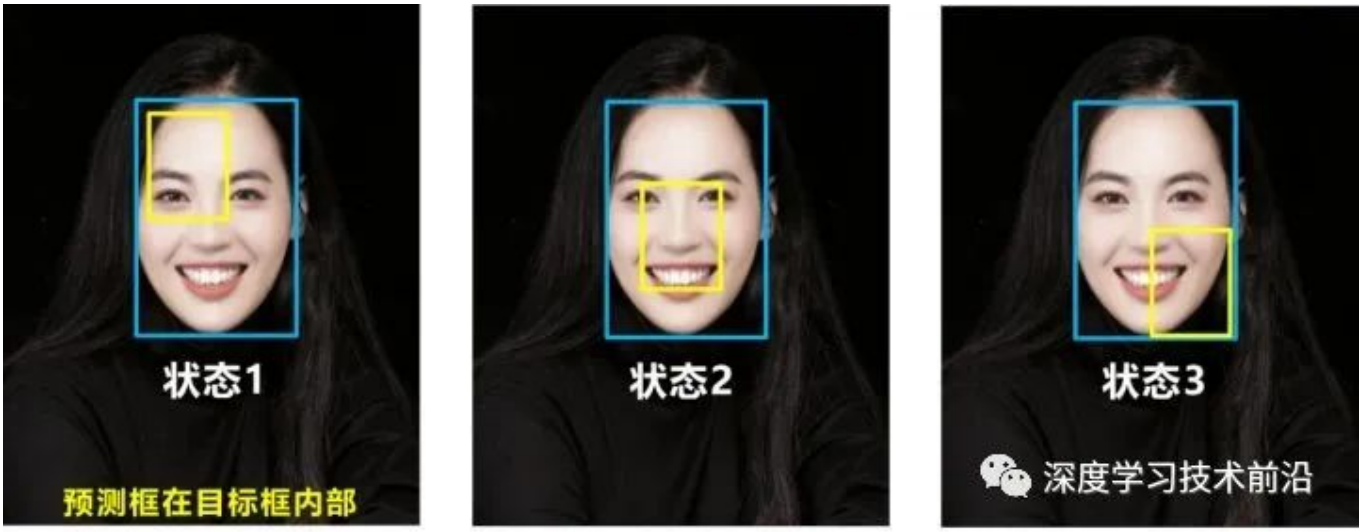
b.GIOU_Loss



可以看到右圖GIOU_Loss中，增加了相交尺度的衡量方式，緩解了單純IOU_Loss時的尷尬。

但為什麼僅僅說緩解呢?

因為還存在一種**不足**:



問題：狀態1、2、3都是預測框在目標框內部且預測框大小一致的情況，這時預測框和目標框的差集都是相同的，因此這三種狀態的**GIOU值**也都是相同的，這時GIOU退化成了IOU，無法區分相對位置關係。
基於這個問題，**2020年**的AAAI又提出了**DIOU_Loss**。

c.DIOU_Loss

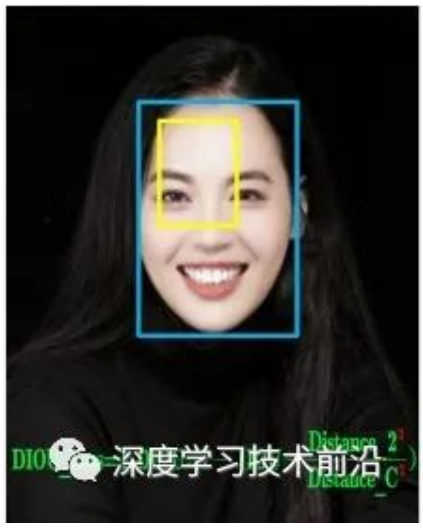
好的目標框回歸函數應該考慮三個重要幾何因素：**重疊面積、中心點距離，長寬比。**

針對IOU和GIOU存在的問題，作者從兩個方面進行考慮

一：如何最小化預測框和目標框之間的歸一化距離?

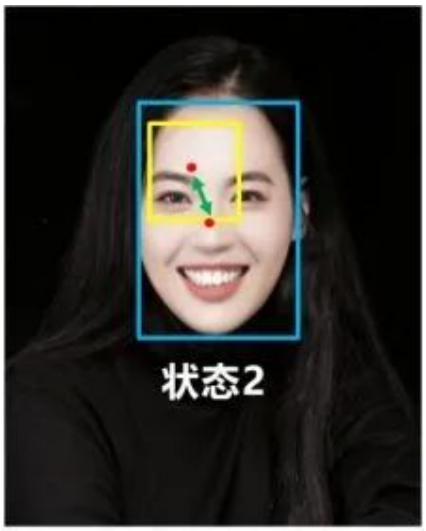
二：如何在預測框和目標框重疊時，回歸的更準確?

針對第一個問題，提出了DIOU_Loss（Distance_IOU_Loss）



DIOU_Loss考慮了**重疊面積**和**中心點距離**，當目標框包裹預測框的時候，直接度量2個框的距離，因此DIOU_Loss收斂的更快。

但就像前面好的目標框回歸函數所說的，沒有考慮到長寬比。



比如上面三種情況，目標框包裹預測框，本來DIOU_Loss可以起作用。

但預測框的中心點的位置都是一樣的，因此按照DIOU_Loss的計算公式，三者的值都是相同的。

針對這個問題，又提出了CIOU_Loss，不對不說，科學總是在解決問題中，不斷進步！！

d.CIOU_Loss

CIOU_Loss和DIOU_Loss前面的公式都是一樣的，不過在此基礎上還增加了一個影響因子，將預測框和目標框的長寬比都考慮了進去。

$$CIOU\_Loss=1-CIOU=1-\left(IOU-\frac{Distance\_2^2}{Distance\_C^2}-\frac{v^2}{(1-IOU)+v}\right)$$

其中v是衡量長寬比一致性的參數，我們也可以定義為：

$$v=\frac{4}{\pi^2}\left(\arctan\frac{w^{gt}}{h^{gt}}-\arctan\frac{w^{p}}{h^{p}}\right)^2$$

這樣CIOU_Loss就將目標框回歸函數應該考慮三個重要幾何因素：重疊面積、中心點距離，長寬比全都考慮進去了。

再來綜合的看下各個Loss函數的不同點：

IOU_Loss： 主要考慮檢測框和目標框重疊面積。

GIOU_Loss： 在IOU的基礎上，解決邊界框不重合時的問題。

DIOU_Loss： 在IOU和GIOU的基礎上，考慮邊界框中心點距離的信息。

CIOU_Loss： 在DIOU的基礎上，考慮邊界框寬高比的尺度信息。

Yolov4中採用了**CIOU_Loss**的回歸方式，使得預測框回歸的**速度和精度**更高一些。

（2）DIOU_nms

Nms主要用於預測框的篩選，常用的目標檢測算法中，一般採用普通的nms的方式，Yolov4則藉鑑上面D/CIOU loss的論文：arxiv.org/pdf/1911.0828

將其中計算IOU的部分替換成DIOU的方式：

再來看下實際的案例

在上圖重疊的摩托車檢測中，中間的摩托車因為考慮邊界框中心點的位置信息，也可以回歸出來。

因此在重疊目標的檢測中，**DIOU_nms的效果優於傳統的nms**。

注意：有讀者會有疑問，這里為什麼不用CIOU_nms，而用DIOU_nms?

答：因為前面講到的CIOU_loss，是在DIOU_loss的基礎上，添加的影響因子，包含groundtruth標註框的信息，在訓練時用於回歸。

但在測試過程中，並沒有groundtruth的信息，不用考慮影響因子，因此直接用DIOU_nms即可。

總體來說， YOLOv4的論文稱的上良心之作，將近幾年關於深度學習領域最新研究的tricks移植到Yolov4中做驗證測試，將Yolov3的精度提高了不少。

雖然沒有全新的創新，但很多改進之處都值得借鑒，借用Yolov4作者的總結。

Yolov4 主要帶來了3 點新貢獻：

（1）提出了一種高效而強大的目標檢測模型，使用1080Ti 或2080Ti 就能訓練出超快、準確的目標檢測器。

（2）在檢測器訓練過程中，驗證了最先進的一些研究成果對目標檢測器的影響。

（3）改進了SOTA 方法，使其更有效、更適合單GPU 訓練。

5.YoloV4相關代碼

5.1 python代碼

代碼地址：https://github.com/Tianxiaomo/pytorch-Yolov4

作者的訓練和測試推理代碼都已經完成

5.2 C++代碼

Yolov4作者Alexey的代碼，俄羅斯的大神，應該是個獨立研究員，更新算法的頻繁程度令人佩服。

在Yolov3作者Joseph Redmon宣布停止更新Yolo算法之後，Alexey憑藉對於Yolov3算法的不斷探索研究，贏得了Yolov3作者的認可，發布了Yolov4。

代碼地址：https://github.com/AlexeyAB/darknet