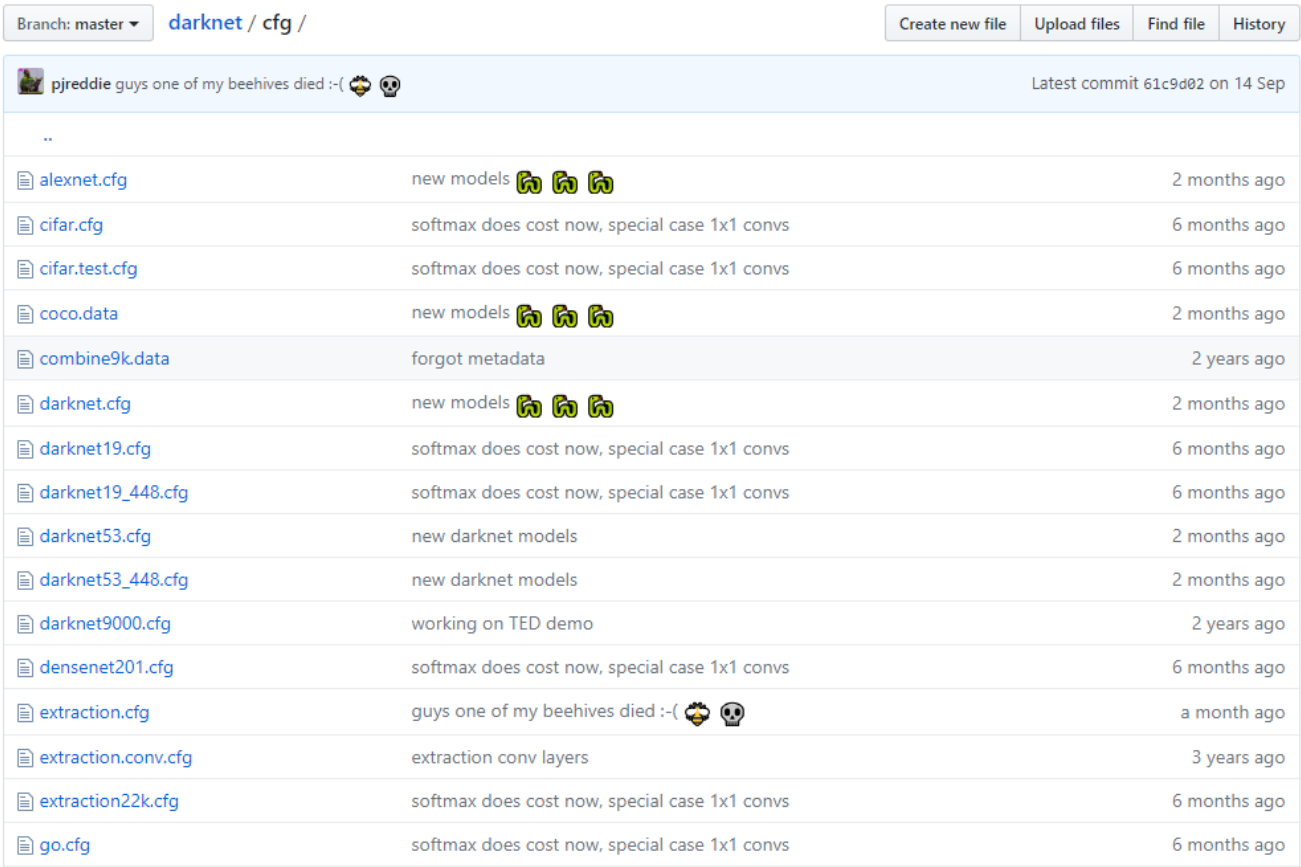
深度學習-物件偵測YOLOv1、YOLOv2和YOLOv3 cfg 檔解讀(一)

Tommy Huang
Oct 23, 2018 · 12 min read

> *Note: 此篇**沒有針對YOLO CNN結構的參數([convolutional]、[maxpool]、[shortcut]、[route]、[reorg])進行說明**，如果想了解請看這篇**深度學習-物件偵測YOLOv1、YOLOv2和YOLOv3 cfg 檔解讀(二)**，這篇純粹說明一些模型學習參數等。(其實是寫完這篇才想到這些參數忘記寫)*

此篇主要是說明YOLOv1~v3 cfg檔解讀，主要是參考YOLO開源連結檔和相關論文。進入github開源cfg資料夾內放的就是YOLO結構的檔案，如下:



Branch: master ▾ darknet / cfg / Create new file | Upload files | Find file | History

pjreddie guys one of my beehives died :-(Latest commit 61c9d02 on 14 Sep

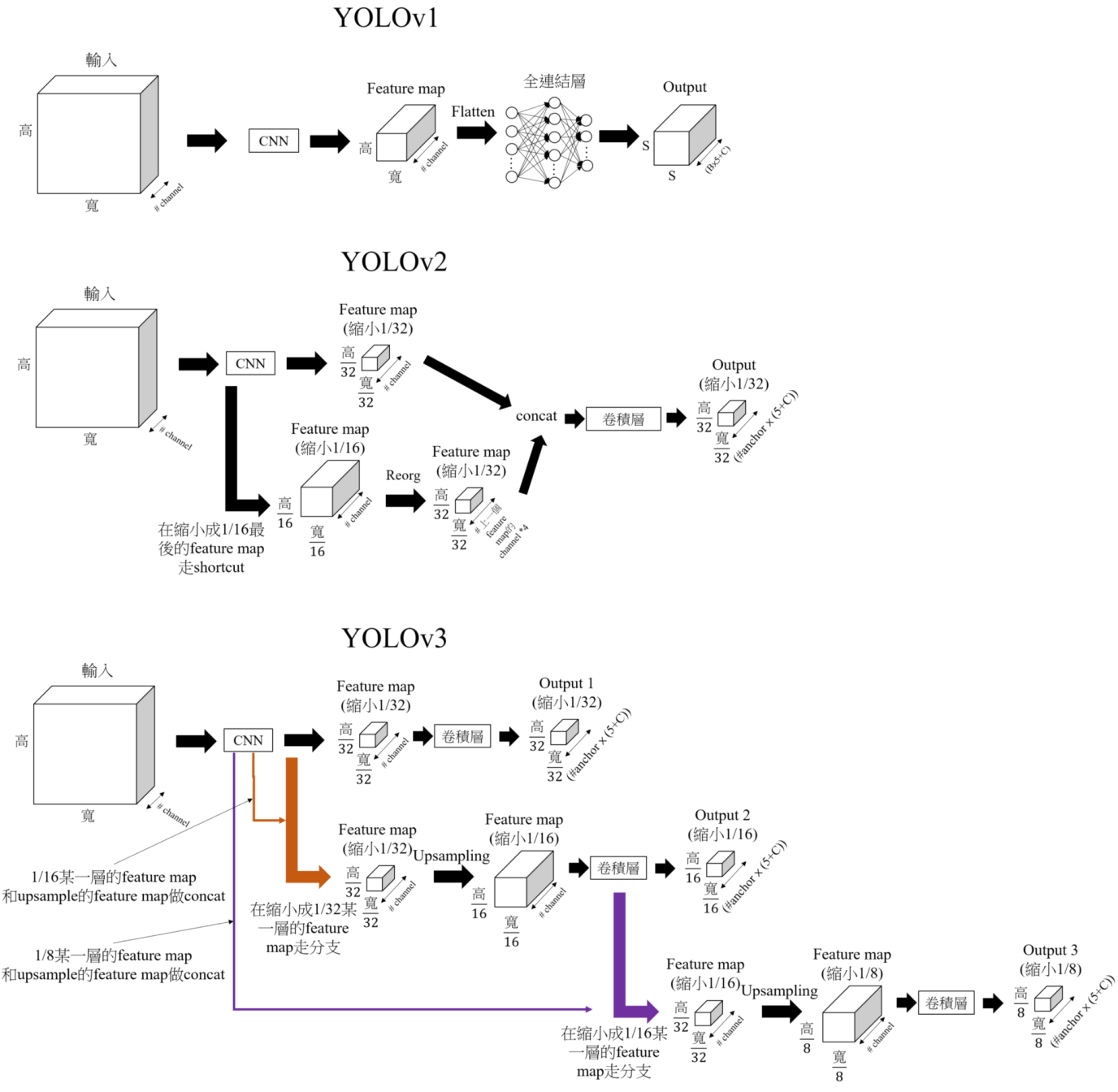
..		
alexnet.cfg	new models	2 months ago
cifar.cfg	softmax does cost now, special case 1x1 convs	6 months ago
cifar.test.cfg	softmax does cost now, special case 1x1 convs	6 months ago
coco.data	new models	2 months ago
combine9k.data	forgot metadata	2 years ago
darknet.cfg	new models	2 months ago
darknet19.cfg	softmax does cost now, special case 1x1 convs	6 months ago
darknet19_448.cfg	softmax does cost now, special case 1x1 convs	6 months ago
darknet53.cfg	new darknet models	2 months ago
darknet53_448.cfg	new darknet models	2 months ago
darknet9000.cfg	working on TED demo	2 years ago
densenet201.cfg	softmax does cost now, special case 1x1 convs	6 months ago
extraction.cfg	guys one of my beehives died :-(	a month ago
extraction.conv.cfg	extraction conv layers	3 years ago
extraction22k.cfg	softmax does cost now, special case 1x1 convs	6 months ago
go.cfg	softmax does cost now, special case 1x1 convs	6 months ago

我們主要看yolov2.cfg, yolov3.cfg 這個跟檔名後面帶著-xxx，例如: yolov2-voc.cfg, yolov3-voc.cfg就是不同的database訓練的cfg檔案，基本上差別在輸出層(分類20類和

80類，輸出層輸出一定不同)而已。沒有帶- xxx的基本上是COCO database總類別數是80類。帶有-voc是跑VOC database類別數是20類。

> *Note: yolov1.cfg是跑VOC database這個跟上面講的不太一樣。*
> *如果你不想瞭解算法，但想拿YOLOv1~v3直接跑你的資料，你只要注意最後輸出層的寫法即可*

下圖為YOLOv1、YOLOv2和YOLOv3的網路結構，CNN(darnet-19, darnet-53之類)的細部我都沒有提到，此圖主要是要說明三個版本模型的差異，但坦白說darnet-19和53只是作者用卷積層和池化層(53加入residual層)組起來的結構，結構不難可以稍微仔細研讀。
此圖對後面講解cfg檔的輸出層output數字是很重要的，可以對照此圖來看。



. . .

本篇cfg檔說明章節結構如下:

- **YOLOv1～YOLOv3 cfg檔學習參數說明**
- **YOLOv1～v3 cfg檔最後面的參數說明**
  YOLOv1～v3類別數的輸出設定與關係
  YOLOv1～v3最後一層其他參數說明
- **Data augmentation參數說明**
- **YOLOv2和YOLOv3 anchor設計重要性**

. . .

YOLOv1~YOLOv3 cfg檔學習參數說明

主要學習參數都在最前面的[net]結構內，三個版本寫法都一樣。

```
momentum=0.9
decay=0.0005
learning_rate=0.0005
policy=steps
steps=200,400,600,20000,30000
scales=2.5,2,2,.1,.1
max_batches = 40000
```

momentum: 利用momentum這個optimizer來訓練 (細節看: 機器/深度學習-基礎數學(三):梯度最佳解相關算法(gradient descent optimization algorithms))。

decay : decay是weight decay的參數，主要是用在參數正規化權重避免overfitting，YOLO系列是在卷積層內kernel map上用L-2 regularization。

learning_rate: optimizer學習參數 (細節看: 機器/深度學習-基礎數學(二):梯度下降法(gradient descent))

max_batches: 模型最大迭代次數。

policy=steps 這個參數是用來調整學習率的策略(在機器/深度學習-基礎數學(二):梯度下降法(gradient descent)內有說明學習率的大小會影響找解的問題，所以用不同策略解決學習率在不同時間上大小要不同)，包含CONSTANT, STEP, EXP, POLY，STEPS, SIG, RANDOM。

steps : 這邊就是當迭代在第幾次時，學習率會發生變化。

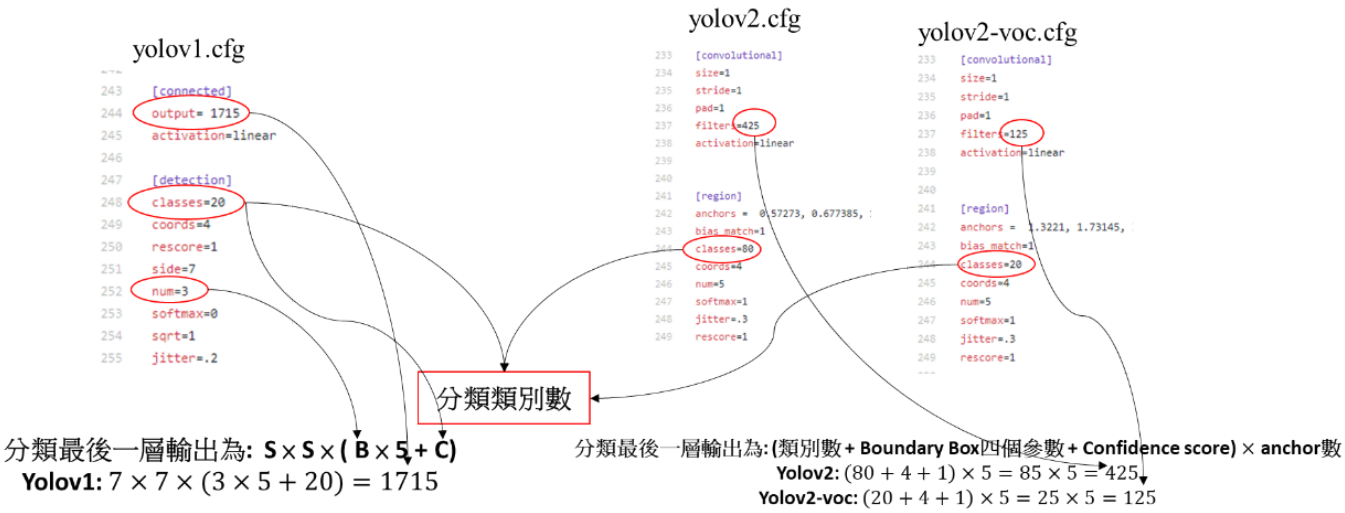
scales : 和steps是一組的，所以數量和steps是一樣的，這個參數就是在第幾次時，學習率會發生變化比例。

. . .

YOLOv1~v3 cfg檔最後面的參數說明

YOLOv1~v3類別數的輸出設定與關係

YOLOv3輸出跟YOLOv2方法一樣，所以只看YOLOv1和YOLOv2來說明輸出層參數。



YOLOv1和YOLOv2, YOLOv3最大的差異在最後一層，YOLOv1最後一層是全連結層輸出，YOLOv2和YOLOv3引進Faster RCNN的anchor設計所以最後一層是卷積層輸出。

最後判斷偵測的結果在YOLOv1為[detection]、YOLOv2為[region]和YOLOv3為[yolo]，很神奇的三個版本用了三個方式當作最後的輸出寫法，後面開始要寫細節部分。

YOLOv1結構downscaling 6次(縮小1/64)，YOLOv1最後一層是全連結層，所以最後一層輸出是一個array(上圖yolov1的第244行output=1715)，所以此例是1715個輸出，這1715個輸出轉成feature map來看就是7×7×(3×5+20)的結構，YOLOv1輸出實際結構如果寫成參數S×S×(B×5+C)。
S等於input解析度(Input解析度是448*448)downscaling完的結果，448 downscaling 6次(縮小1/64)= $448/(2^6)=7$
B等於第252行寫的num=3，一個格子預測3個Boundary box。
5等於第249行寫的coords=4 (每個Boundary box中心的x座標和y座標、每個Boundary box的長、寬) + 1個confidence score。
C等於第248行寫的classes=20 (最後判斷有20個類別)。

YOLOv2結構downscaling 5次(縮小1/32)，但不同於v1，YOLOv2之後是只有卷積輸出當作後的結果，所以Input解析度是416*416，所以最後輸出是一個
13×13×((5+20) ×5)= 13×13×125 (上圖yolov2的第237行filter=125)
YOLOV2輸出也是類似S×S×(B×5+C)的結構:
S等於input解析度(Input解析度是448*448)downscaling完的結果，448 downscaling 5次= $416/(2^5)=13$
B等於第246行寫的num=5，一個格子預測5個anchors。
5等於第245行寫的coords=4 (每個anchor 的Boundary box中心的x座標和y座標、每個Boundary box的長、寬) +這個anchor的 1個confidence score。
C等於第244行寫的classes=20 (最後判斷有20個類別)。

YOLOv3結構比較特殊，除了用了darknet-53當作feature extractor外，他引用「多尺度偵測」方式(第一張圖有圖示)，從(縮小1/32、縮小1/16和縮小1/8)三個尺度分別去做偵測，所以YOLOv3-voc.cfg檔案裡面有三個[yolo層]。
假設Input解析度是416*416，輸出也是類似S×S×(B×5+C)的結構，每個尺度各帶入3個anchor
縮小1/32 : 13×13×((5+20) ×3)= 13×13×75 (第605行寫的filters=75)
縮小1/16 : 26×26×((5+20) ×3)= 26×26×75 (第689行寫的filters=75)

縮小1/8 : 52×52×((5+20) ×3)= 52×52×75 (第773行寫的filters=75)
YOLOv3的num=9，雖然說有9個anchor，但實際用到幾個anchor是看[yolo]下面的mask，mask=0, 1, 2 代表用這個尺度用的anchor是用[yolo]下面的anchors的前三組(一組兩個數字)，mask=3, 4, 5 代表用這個尺度用的anchor是用[yolo]下面的anchors的第4、5、6組(一組兩個數字)。

YOLOv1~v3最後一層其他參數說明

YOLOv1和YOLOv2有

```
object_scale=5
noobject_scale=1
class_scale=1
coord_scale=1
```

這四個是YOLOv1論文中提到cost function的權重，細節可以看論文「You Only Look Once: Unified, Real-Time Object Detection」或「深度學習-物件偵測:You Only Look Once (YOLO)」，主要是針對物件偵測結果和非物件偵測結果給予不同權重，這樣模型才能學到物件的特徵，而不是非物件(背景)的特徵。這四個參數很神奇在YOLOv3沒有出現，我個人猜測是作者覺得這個參數應該不會改所以用default的，就沒有在YOLOv3.cfg檔去寫。

YOLOv2和YOLOv3有

```
#YOLOV2
absolute=1 # 目前看不懂這個參數在幹嘛
thresh = .6 #這個參數跟YOLOv3中的ignore_thresh是一樣作用的，用來決定物件是否要被納入cost function計算。若best IoU大於此參數，則此IoU誤差不會納入cost function中。
random=1

#YOLOV3

ignore_thresh = .5 (同上說明)
truth_thresh = 1 # 目前看不懂這個參數在幹嘛
random=1
```

random=1代表每10次迭代輸入影像會隨機縮放到320*320–608*608。

. . .

Data augmentation參數說明

Data augmentation參數在YOLOv1～v3寫法都一樣沒變，我這邊舉YOLOv2的cfg來看

三個版本的最後一層都有jitter 這個參數，這個參數如同中文叫做「抖動」，主要是利用這個方法(crop, flip)產生更多資料抑制overfitting。

```
Jitter=0.3
```

代表0～0.3比例隨機裁減圖片。

[net]層有angle(角度), saturation(飽和度), exposure(曝光度), hue(色調)

```
angle=0
saturation = 1.5
exposure = 1.5
hue=.1
```

這些參數就是用來隨機產生新的資料用來訓練模型。
angle(角度): 圖片角度變化，單位是度， angle=10，就是隨機生成-10～10度旋轉的圖片。

saturation(飽和度)和exposure(曝光度): 代表飽和度和曝光度設定1.5代表用飽和度和曝光度進行1～1.5倍隨機生成圖片。

hue(色調): 色調設定為0.1代表在色調隨機變化-0.1～0.1生成圖片。

. . .

YOLOv2和YOLOv3的anchor設計的問題

YOLO作者在github上針對cfg在anchor設計解釋，YOLOv2的anchor參數是相對的大小，也就是在縮小1/32後的feature map上對應的大小，所以YOLOv2 cfg檔寫的數字要對到實際圖上，需乘上32，才會是實際anchor的大小。
而YOLOv3這個版本，anchor參數直接改成實際圖上的pixel大小。
YOLOv2的anchor有小數點(因為是相對的值)，YOLOv3的值都是整數(pixel大小)。

從github內節錄

. . .

YOLOv2和YOLOv3 anchor設計重要性

YOLOv2和YOLOv3的anchor怎麼轉到原始圖上的大小，論文上寫的公式如下:

20181213修正此圖

(Anchor放大縮小的說法在20181213有修正)

所以每個格子預測的物件是從anchor去放大(寬:exp(dw>0), 高:exp(dh>0))和縮小(寬:exp(dw <0) , 高: exp(dh<0))。
如果anchor設計太大，小物件就抓不好(要縮小很多倍)
anchor設計太小大物件就可能抓不好(要放大很多倍)
實際運作上放大太多跟縮小太多都不太合理，比如某個Bbox是Anchor需要放大20倍，所以通常實際運作時這個BBox真的屬於物件的confidence score也相對會低很多。

YOLO作者提供一個功能，用k-means去自動算出你的資料集anchor大小設計多少比較合適，且特殊的一點是作者捨去一般k-means用歐式距離當距離測度，用IoU當作距離測度來執行k-means。

> *NOTE: 很多人都直接用default來跑，結果performance很糟糕，那就要考慮發生default不match你的資料，所以要用k-means重新訓練anchor size。*

. . .

內容不一定完全對，如果錯誤在麻煩幫我指出，感謝。