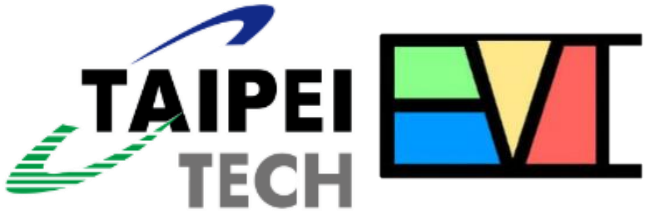


Embedded Vision Intelligent Laboratory

嵌入式智慧影像分析與實境界面
Fall 2021

Instructor：Yen-Lin Chen(陳彥霖), Ph.D.

Professor

Dept. Computer Science and Information Engineering

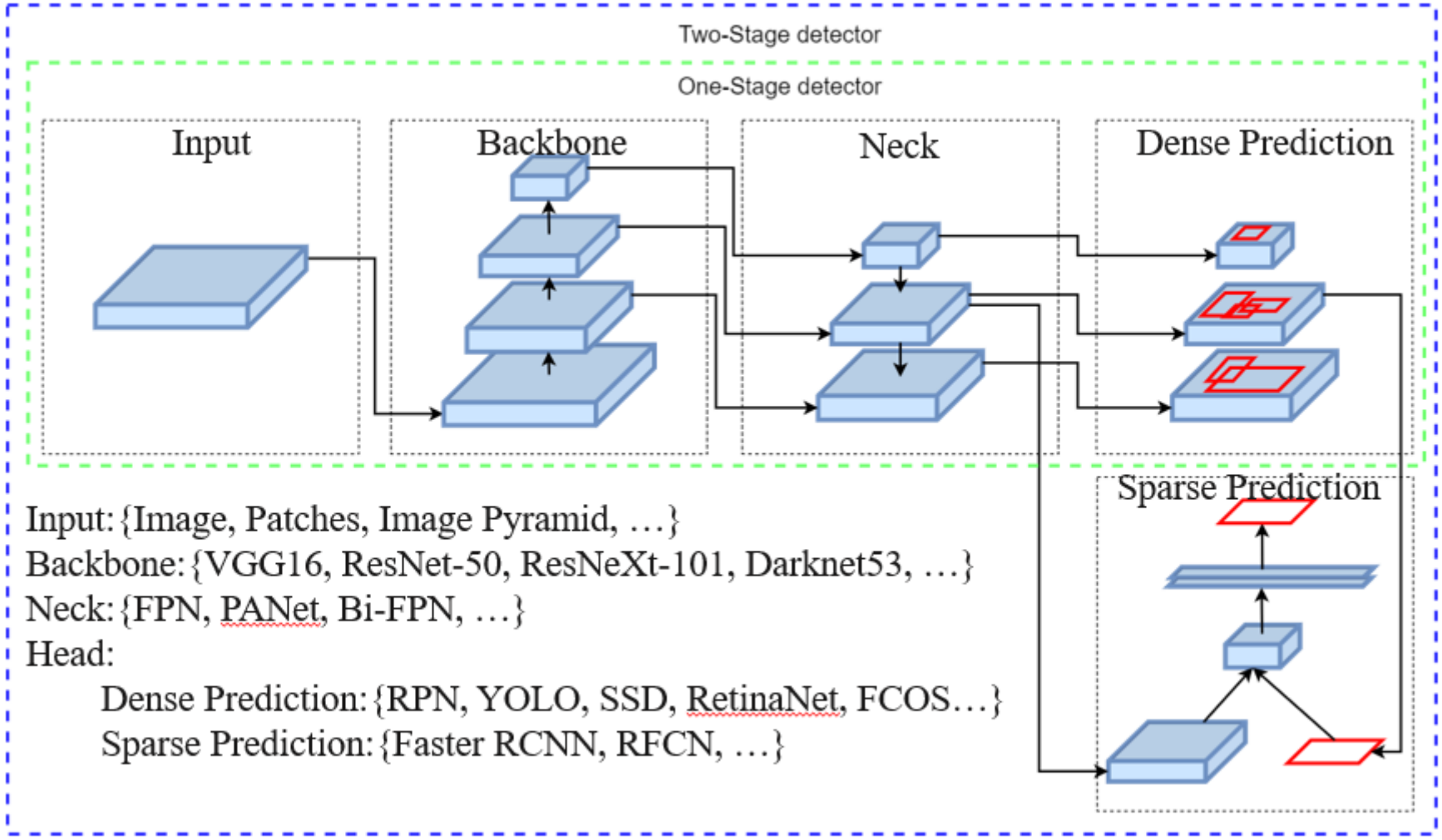
National Taipei University of Technology

Lecture 3

YOLO影像識別深度學習網路模型

One/Two-Stage Detector

One-Stage Detector versus Two-Stage Detector



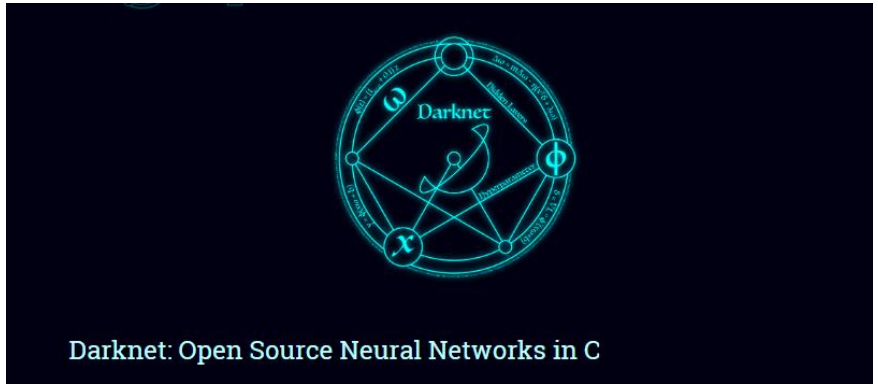
One-Stage Detector versus Two-Stage Detector

- 物件辨識主流的演算法框架大致分為two-stage 與one-stage。代表分別為R-CNN系列、YOLO系列。
- two-stage演算法將分為兩步驟執行，輸入圖像先經過候選框生成網路（例如Faster RCNN中的RPN網路）以檢測物件位置，再經過分類網路以進行物件分類。
- one-stage演算法將則一步驟就完成偵測、分類，輸入圖像只經過一個網路，生成的結果中同時包含位置與類別資訊。
- two-stage與one-stage相比，雖精度高，但是計算量大、運算較慢。

You Only Look Once (YOLO)

You Only Look Once (YOLO)

- YOLO的深度學習框架：Darknet
- Darknet比較冷門，是用C語言及CUDA撰寫，因此計算速度較快。
- Darknet支援CPU與GPU。



- 由於使用C語言撰寫，因此較容易移植到其他平台。
- Darknet與其他框架相比更加輕量化。

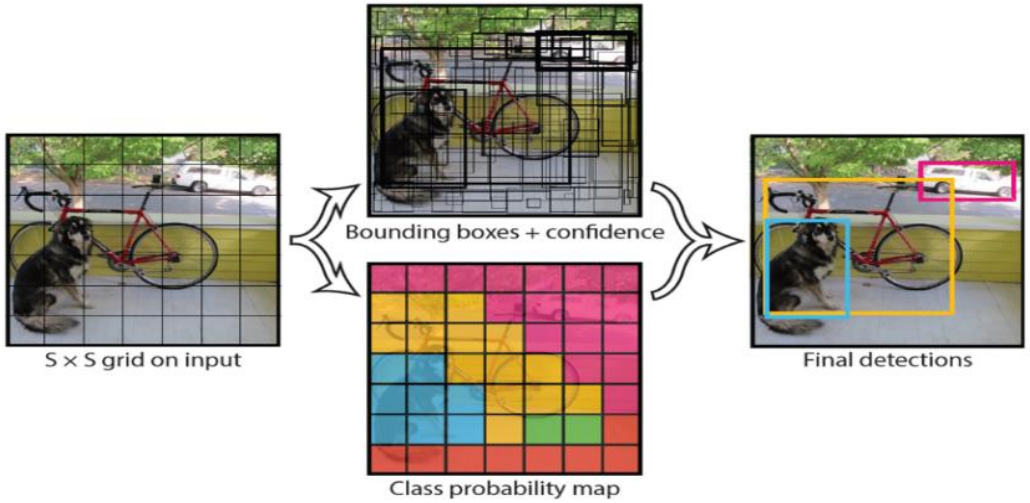
You Only Look Once (YOLOv1)

- YOLO是一個可以一次性預測多個Box位置和類別的卷積網路。
- 實現End-to-End的目標檢測和識別。
- 最大的優勢就是速度快，但會犧牲一些精度。
- YOLO沒有選擇Slide Window或提取Fast-RCNN的方式訓練網路，而是直接選用整張影像來訓練模型，這樣做的好處在於可以更好的區分目標和背景區域。
- 主要的卷積網路是GoogleNet。

You Only Look Once (YOLOv1)

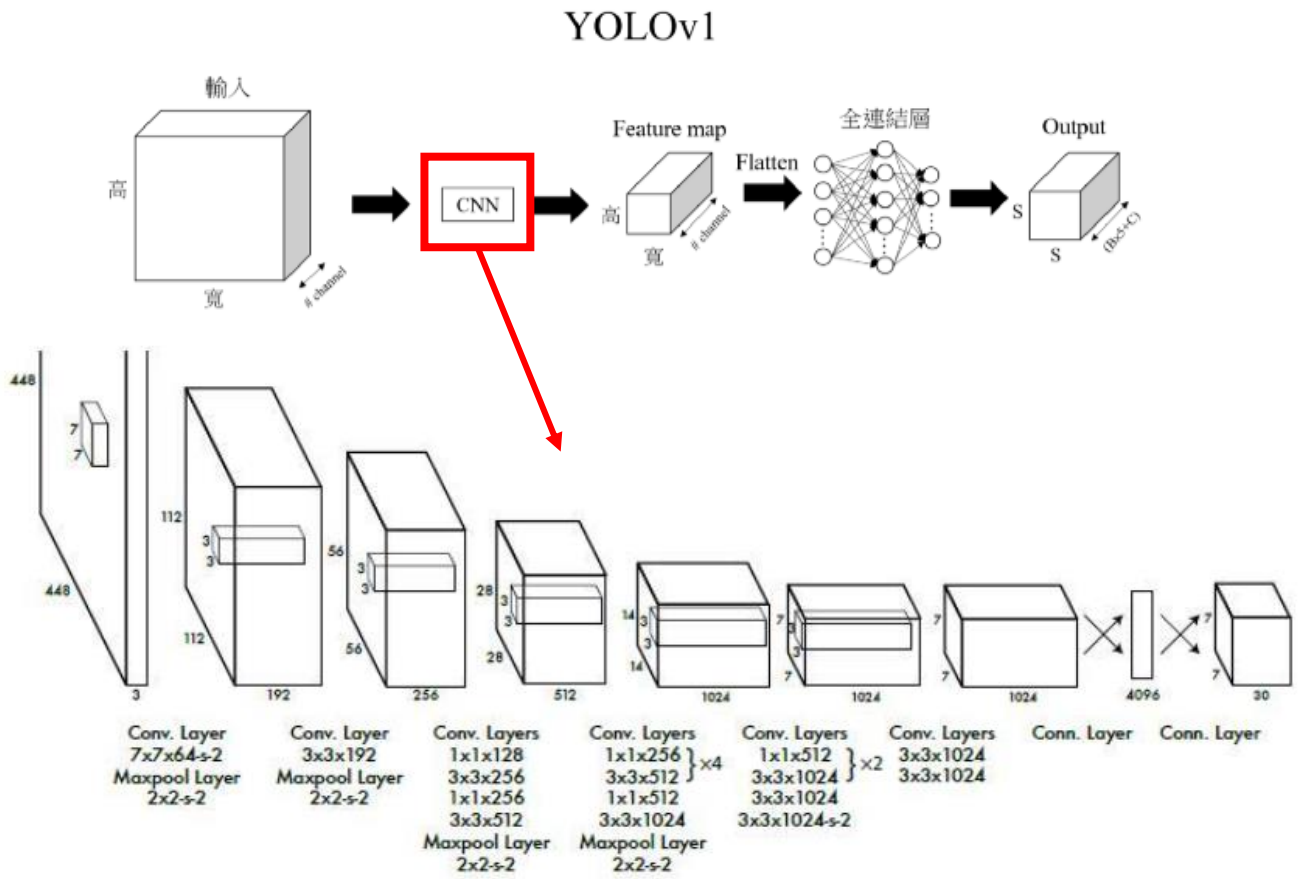
- YOLOv1檢測流程：

  每個影像切成SxS的格子(grid)，如果格子中間有物體則該格子會負責去偵測該物體。每個格子又會預測B個bounding box與其confidence score，其中conf. scores對應bounding box含有物體的信心程度以及該bounding box中的物體的精準度。每個bounding box都有五個預測參數，x, y, w, h, confidence，(x, y)表示物件對grid cell的中心相對位置，而w, h為bounding box長寬，confidence即為IOU (Intersection over union)。



You Only Look Once (YOLOv1)

- YOLOv1網路結構圖



- 網路方面主要採用GoogLeNet，卷積層主要用來提取特徵，全連接層主要用來預測類別機率和坐標。最後的輸出是7x7x30，7x7是grid cell的數量。

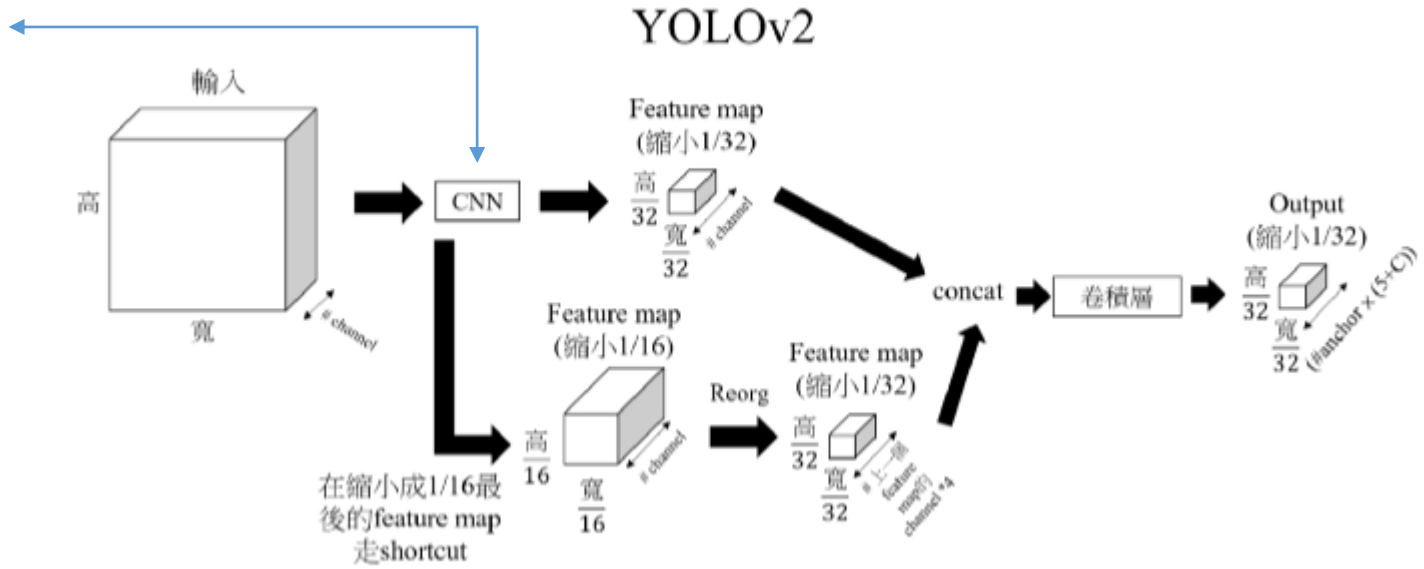
You Only Look Once (YOLOv2)

- 導入Faster RCNN中的anchor box技術，使用K-Means求anchor box的比例，不再直接透過mapping獲得bounding box的座標。
- 使用convolution layer取代fully connection layer。
- 使用batch normalization取代dropout。
- 增加ImageNet pretrain解析度：原本的YOLO網路在預訓練的時候是使用224x224的影像，detect時使用448x448的影像，導致從分類模型切換到檢測模型時，模型還要適應影像解析度的改變。而YOLOv2則將預訓練分成兩個步驟：先輸入224x224的影像從頭開始訓練網路，大概160個epoch，然後將輸入調整到448x448，再訓練10個epoch，因此在detect的時候，輸入448x448的影像就可以順利通過。

You Only Look Once (YOLOv2)

- Darknet 19(左圖)
- YOLOv2網路結構圖(右圖)

Type	Filters	Size/Stride	Output
Convolutional	32	3 × 3	224 × 224
Maxpool		2 × 2/2	112 × 112
Convolutional	64	3 × 3	112 × 112
Maxpool		2 × 2/2	56 × 56
Convolutional	128	3 × 3	56 × 56
Convolutional	64	1 × 1	56 × 56
Convolutional	128	3 × 3	56 × 56
Maxpool		2 × 2/2	28 × 28
Convolutional	256	3 × 3	28 × 28
Convolutional	128	1 × 1	28 × 28
Convolutional	256	3 × 3	28 × 28
Maxpool		2 × 2/2	14 × 14
Convolutional	512	3 × 3	14 × 14
Convolutional	256	1 × 1	14 × 14
Convolutional	512	3 × 3	14 × 14
Convolutional	256	1 × 1	14 × 14
Convolutional	512	3 × 3	14 × 14
Maxpool		2 × 2/2	7 × 7
Convolutional	1024	3 × 3	7 × 7
Convolutional	512	1 × 1	7 × 7
Convolutional	1024	3 × 3	7 × 7
Convolutional	512	1 × 1	7 × 7
Convolutional	1024	3 × 3	7 × 7
Convolutional	1000	1 × 1	7 × 7
Avgpool		Global	1000
Softmax			



You Only Look Once (YOLOv3)

- YOLOv3 makes predictions at 3 different scales (similar to the FPN)
  - Scale1 : 在基礎網路(CNN)後，加上卷積層，為第一次prediction。
  - Scale2 : 在Scale1中倒數第二層卷積層上，然後upsamles兩倍，加上卷積層，為第二次prediction。
  - Scale3 : 跟Scale2相似，第三次prediction 。
- 主要的卷積網路是Darknet-53(類似Resnet101)

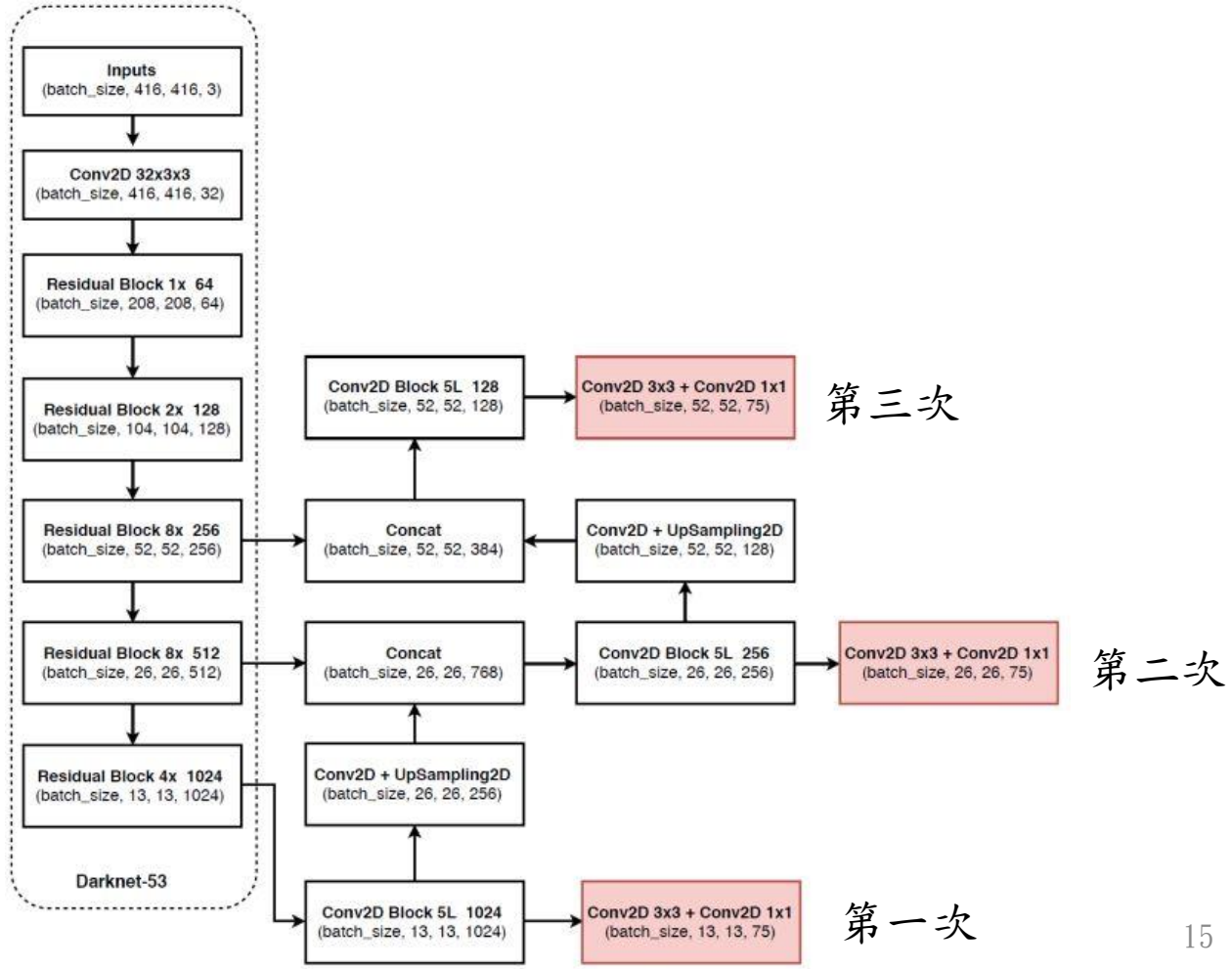
You Only Look Once (YOLOv3)

1. YOLOv3使用resnet網路 (Residual Network)
   新的基底網路為Darknet-53，有53層，隨著網路層數不斷加深(數量級從20~30層到50層)，採用了一般類神經網路加深時常用的 ResNet 結構來解決梯度問題。
2. YOLOv3使用FPN網路 (Feature Pyramid Networks)
   使用FPN多層級預測架構以提升小物體預測能力，特徵層從單層13x13變成了多層13x13、26x26和52x52，單層預測5種bounding box 變成每層3種 bounding box (共9種)，詳見網路結構圖。使用 FPN 的架構可以讓低層較佳的目標位置和高層較佳的語義特徵融合，並且在不同特徵層獨立進行預測，使得小物體檢測改善效果十分明顯。

You Only Look Once (YOLOv3)

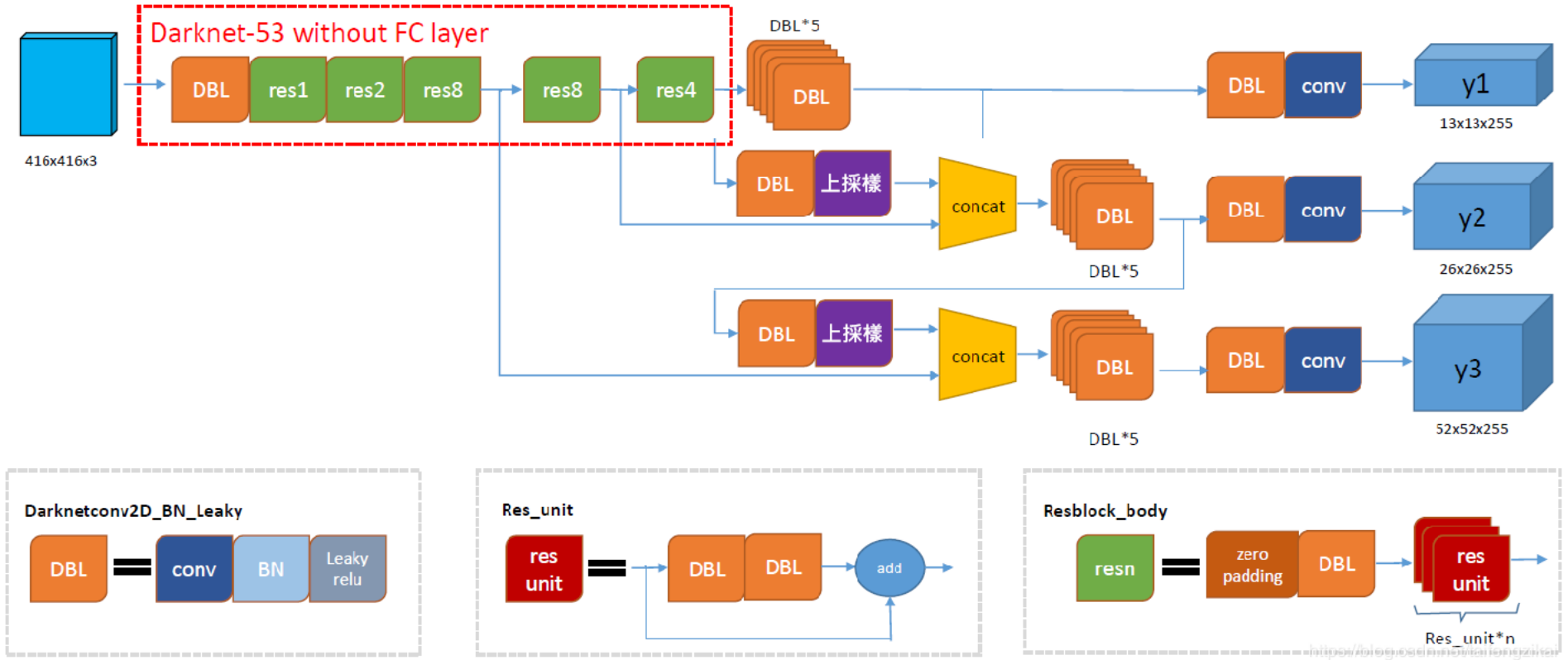
- Darknet53(左圖)
- YOLOv3網路結構圖

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			



You Only Look Once (YOLOv3)

- YOLOv3結構圖



You Only Look Once (YOLOv4)

- YOLOv4 如何實現這麼好的效果？
  - YOLOv4 的基本目標是提高生產系統中神經網絡的運行速度，同時爲並行計算做出最佳化，而不是針對低計算量理論指標（BFLOP）進行最佳化。 YOLOv4 的作者提出了兩種Real Time神經網路：
    - 對於 GPU，研究者在卷積層中使用少量組（1-8 組）：CSPResNeXt50 / CSPDarknet53。
    - 對於 VPU，研究者使用了分組卷積（grouped-convolution），但避免使用 Squeeze-and-excitement （SE）塊。具體而言，它包括以下模型：EfficientNet-lite / MixNet / GhostNet / MobileNetV3。
- YOLOv4 包含以下三部分：
  - 骨幹網路：CSPDarknet53
  - Neck：SPP、PAN
  - Head：YOLOv3

You Only Look Once (YOLOv4)

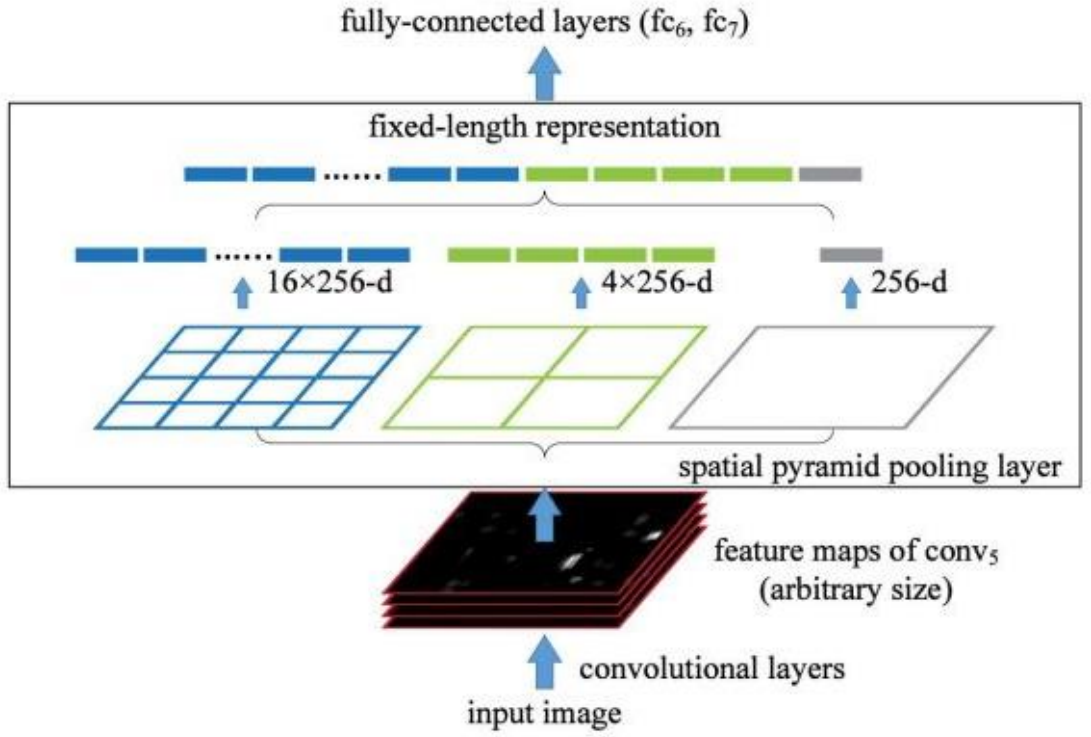
- YOLOv4的骨幹網路採用CSPDarknet53網路結構， CSPDarknet53是以YOLOv3的骨幹網路Darknet53的基礎上產生的Backbone結構。CSPDarknet53主要有三個方面的優點：
  - 優點一：增強CNN的學習能力，使得在輕量化的同時保持準確性。
  - 優點二：降低計算瓶頸。
  - 優點三：降低記憶體成本。
- 最終的模型為：CSPDarkNet53+SPP+PANet(path-aggregation neck)+YOLOv3-head = YOLOv4.

You Only Look Once (YOLOv4)

- CSPDarknet53：結合了於YOLOv3所使用的DarkNet-53與CSPNet (Cross Stage Partial Network)策略的卷積神經網路。它會將feature map切割成兩部分，並在不同階段的層次結構中合併，來讓更多梯度可以通過網路。
- Squeeze-and-Excitation (SE)：Squeeze指的是用Global Average Pooling來將圖片的二維平均化，而Excitation則是使用兩個全連接層和兩個非線性激活函數來學習數據所提供的資訊。這個子架構可以放在不同的網路架構中來獲得不同種類的SENet。

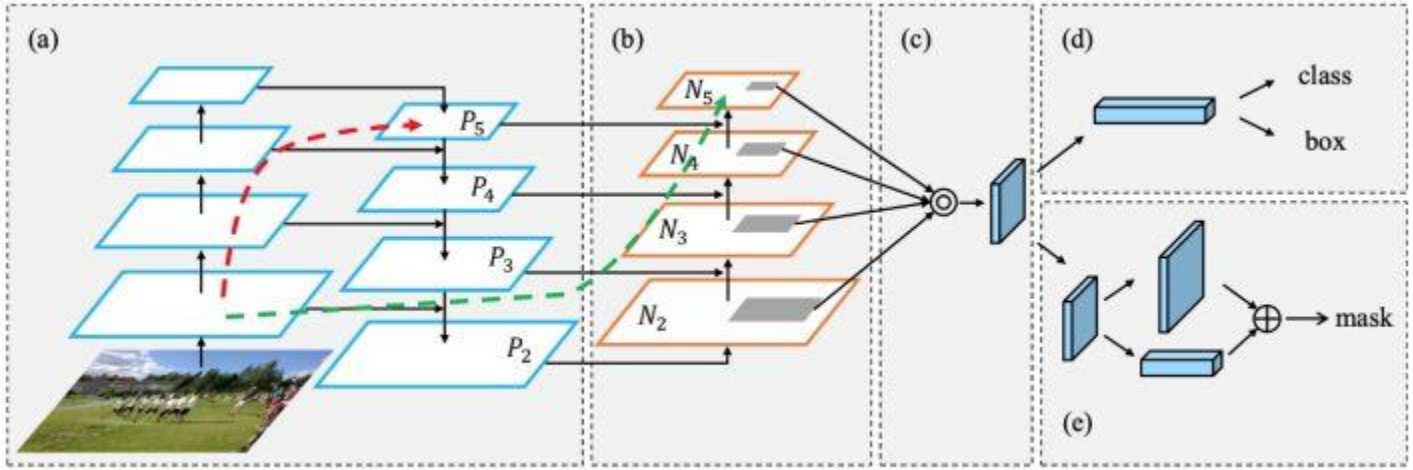
You Only Look Once (YOLOv4)

- SPP：全名為Spatial Pyramid Pooling Layer，它的作用是在最後一個卷積層之後，將feature map以由小到大的許多不同尺寸格子切割，最後組合成一個固定長度的表示法，方便後續全連接層進一步分析資料。



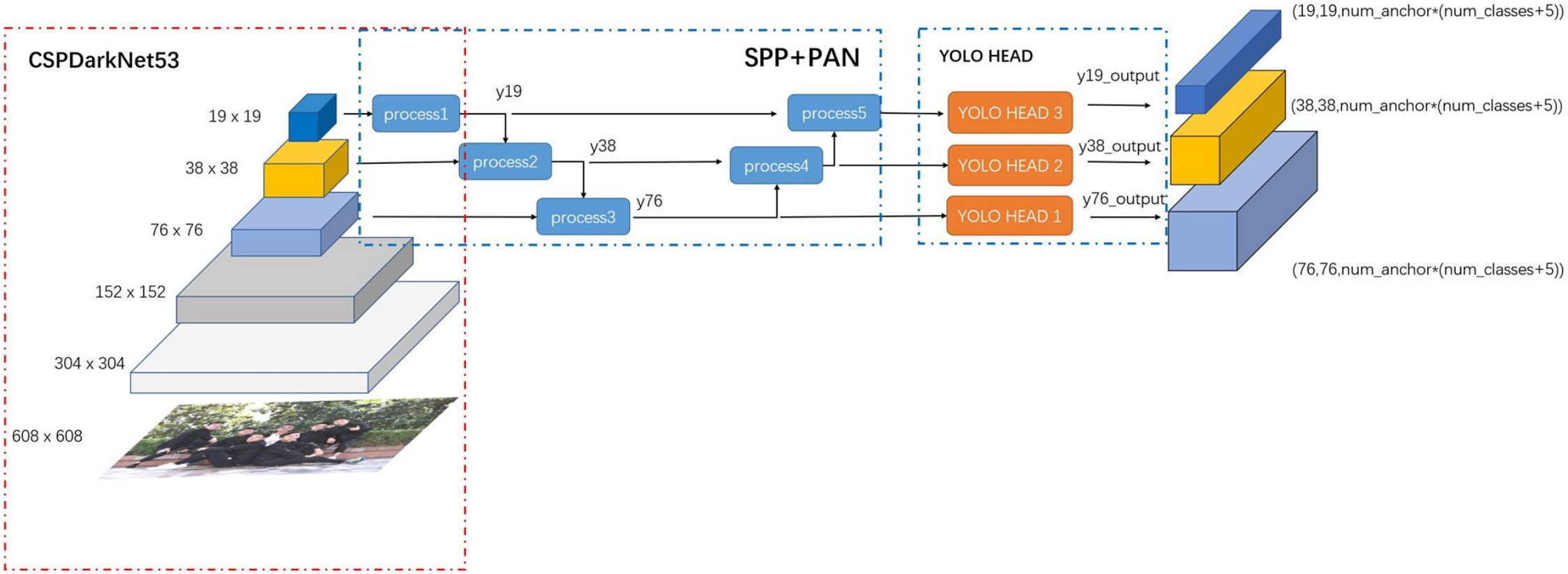
You Only Look Once (YOLOv4)

- PAN：下圖是目標檢測的Path Aggregation Network (PAN)。自底向上的路徑(b)被擴展，使低層資訊更容易傳播到頂層。在先前使用的FPN中，局部空間資訊在紅色箭頭中向上傳遞，雖然在圖中沒有確切顯示，但是紅色的路徑穿過了大約100多層。而PAN引入了一條簡捷的路徑（綠色），只需要大約10層就可以到達頂層N_5。這種捷徑的概念使較細緻的局部資訊也可在頂層被使用。



You Only Look Once (YOLOv4)

YOLOv4結構圖

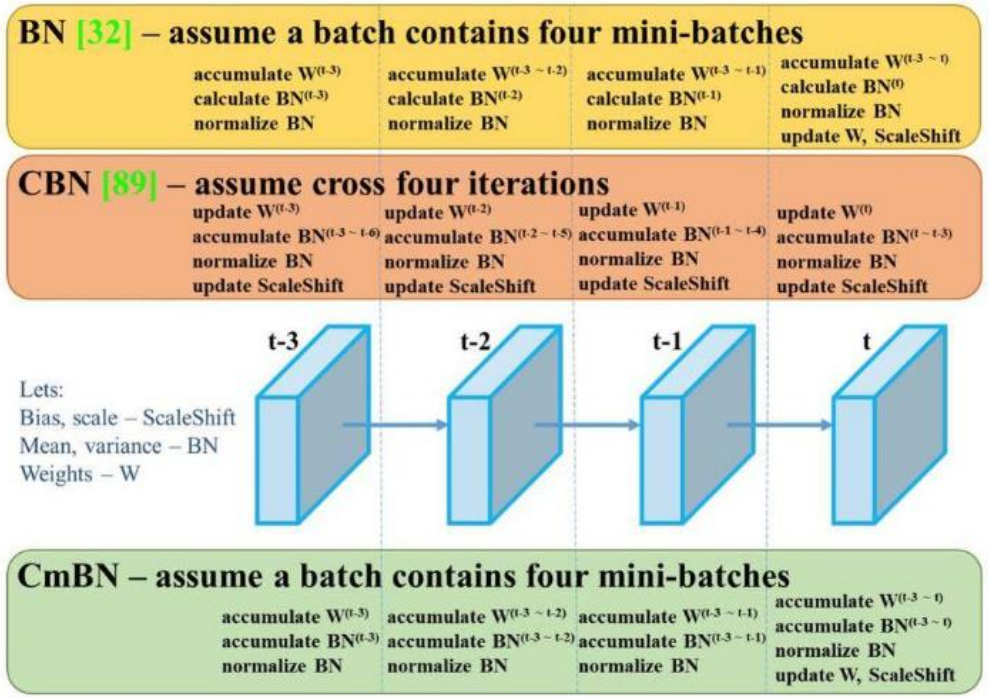


You Only Look Once (YOLOv4)

- 其中，YOLOv4提出新型數據增強方法Mosaic:
- 新型數據增強方法Mosaic混合了4張訓練影像，而CutMix只混合了兩張輸入影像，具體如下圖所示：

You Only Look Once (YOLOv4)

- 自對抗訓練（SAT）也是一種新的數據增強方法，它包括兩個階段。第一個階段中，神經網路更改原始影像；第二階段中，訓練神經網路以正常方式在修改後的影像上執行目標檢測任務。CmBN 是 CBN 的改進版，它僅收集單個批次內 mini-batch 之間的統計數據。



You Only Look Once (YOLOv4)

- 以下4個部分對YOLOv4的創新之處進行講解：
  - 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骨幹網路：將各種新的方式結合起來，包括：CSPDarknet53、Mish激勵函數、Dropblock
  - Neck：目標檢測網路在BackBone和最後的輸出層之間往往會插入一些層，比如YOLOv4中的SPP模組、FPN+PAN結構
  - Prediction：輸出層的anchor box機制和YOLOv3相同，主要改進的是訓練時的損失函數CIOU_Loss，以及將預測框篩選的nms變為DIOU_nms

You Only Look Once (YOLOv4)

- IoU：Intersectoin over Union，即為聯集分之交集，是計算物件偵測的Bounding Box與Ground Truth之間誤差的常用方法。
- CIoU Loss：CIoU全名為Complete IoU，而CIoU Loss公式為

$$L_{CIoU} = 1 - IoU + \frac{\rho^2(b, b^{gt})}{c^2} + \alpha v, \alpha = \frac{v}{(1 - IoU) + v}$$

其中b與b^{gt}代表預測框與實際物件框的中心點，ρ代表兩點的歐式距離函數，c表示能包含預測框與實際物件框的最小框的對角線距離，α表示的是權重函數，而v用來計算長寬比的相似性。

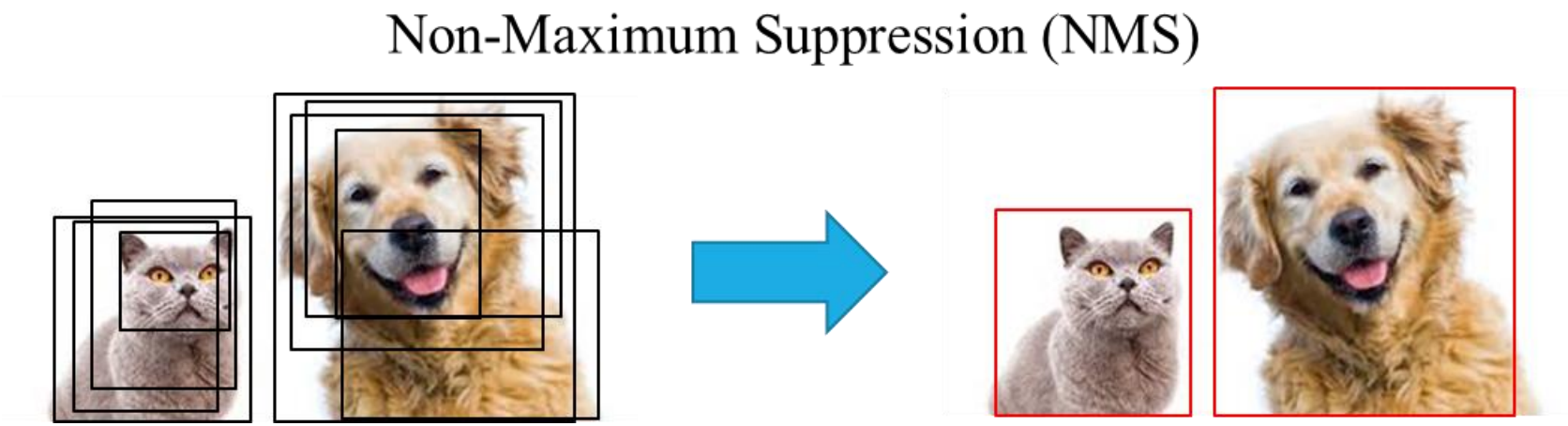
- DIoU：Distance IoU，公式為

$$DIoU = IoU - \frac{\rho^2(b, b^{gt})}{c^2}$$

其中b與b^{gt}代表預測框與實際物件框的中心點，ρ代表兩點的歐式距離函數，c表示能包含預測框與實際物件框的最小框的對角線距離。

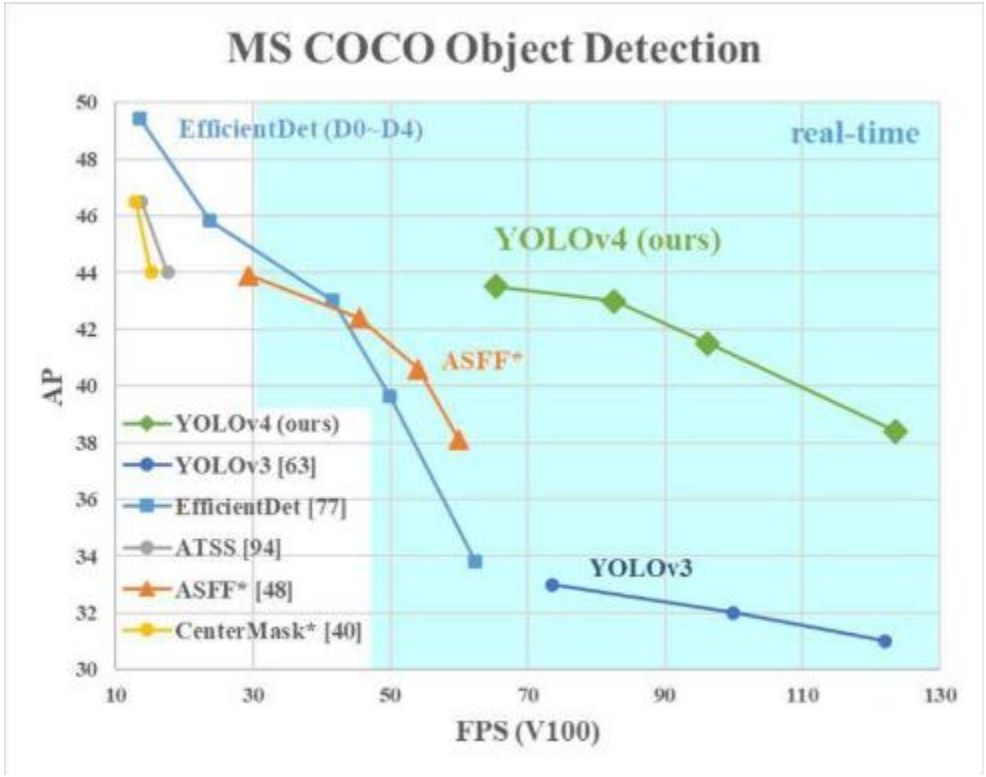
You Only Look Once (YOLOv4)

- NMS：Non-Maximum Suppression。在影像物件偵測領域上，都是先會選出物件候選人，然後在物件候選人中判斷是不是物件，但有可能一個物件被很多候選框給選到（如下圖左）。因此NMS會先找出信心分數最高的Bounding Box，並與其他Bounding Box計算IoU，若IoU大於所設定的門檻值，就把該Bounding Box的信心分數設為0，反覆多次，來刪除其他重複的Bounding Box。



You Only Look Once (YOLOv4)

- YOLOv4 在達到與 EfficientDet 同等性能的情況下，速度是 EfficientDet 的二倍。此外，與 YOLOv3 相比，新版本的 AP 和 FPS 分別提高了 10% 和 12%。



Fast R-CNN、Faster R-CNN與YOLO比較

Real-Time Detectors	Train	mAP	FPS
100Hz DPM [30]	2007	16.0	100
30Hz DPM [30]	2007	26.1	30
Fast YOLO	2007+2012	52.7	**155**
YOLO	2007+2012	**63.4**	45
Less Than Real-Time			
Fastest DPM [37]	2007	30.4	15
R-CNN Minus R [20]	2007	53.5	6
Fast R-CNN [14]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[27]	2007+2012	73.2	7
Faster R-CNN ZF [27]	2007+2012	62.1	18

Ref: You Only Look Once: Unified, Real-Time Object Detection, Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi, arXiv:1506.02640 https://arxiv.org/abs/1506.02640

使用Yolo v4

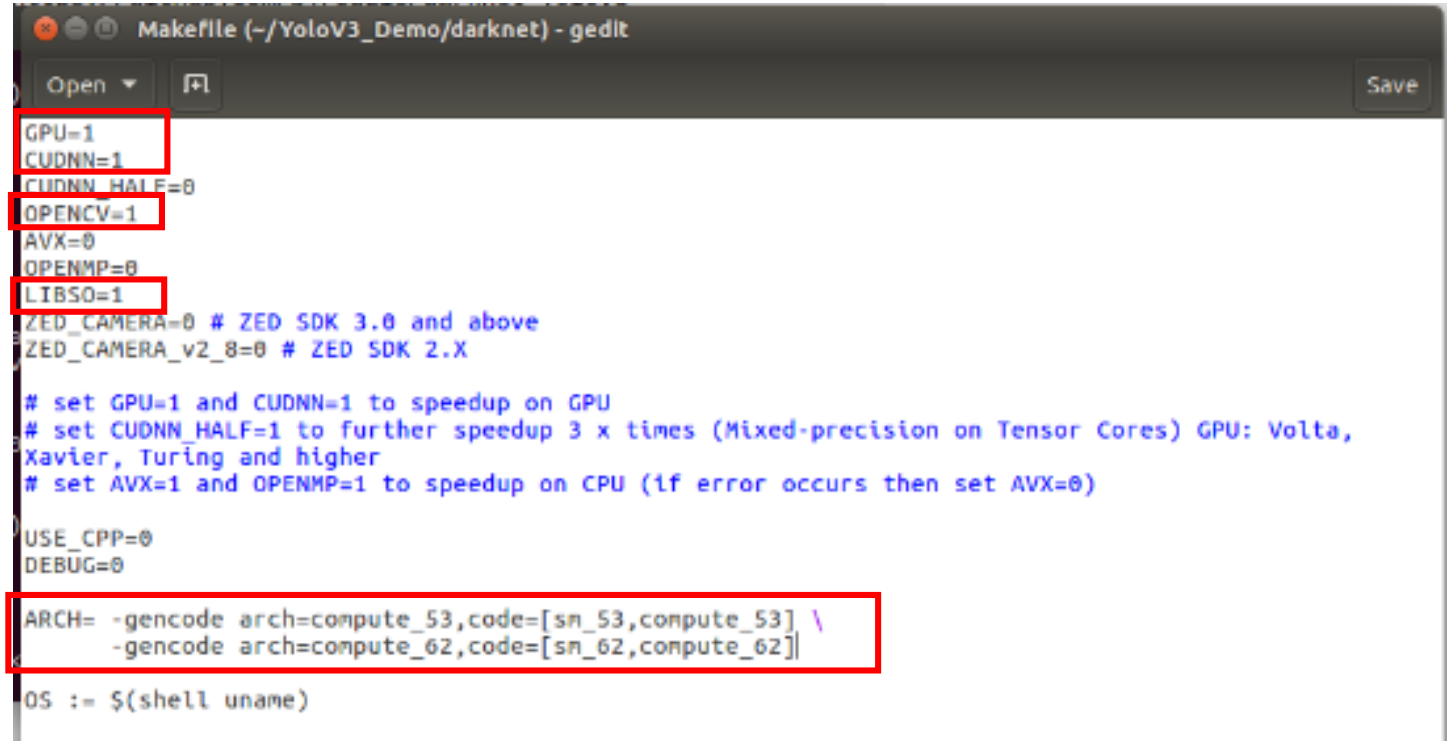
執行YOLOv4

- 將專案clone出來並切換至darknet目錄
- git clone https://github.com/AlexeyAB/darknet.git

```
evil1323@evil1323:~$ git clone https://github.com/AlexeyAB/darknet.git
fatal: destination path 'darknet' already exists and is not an empty directory.
evil1323@evil1323:~$ git clone https://github.com/AlexeyAB/darknet.git
Cloning into 'darknet'...
remote: Enumerating objects: 14291, done.
remote: Total 14291 (delta 0), reused 0 (delta 0), pack-reused 14291
Receiving objects: 100% (14291/14291), 12.84 MiB | 283.00 KiB/s, done.
Resolving deltas: 100% (9751/9751), done.
Checking connectivity... done.
evil1323@evil1323:~$ ls
3.4.7.zip                          Desktop     examples.desktop  Public
anaconda3                          d_labelImg  f_labelimg        pytorch-YOLOv4
Anaconda3-2019.10-Linux-x86_64.sh  dlib        Music             Templates
darknet                            Documents   opencv-3.4.7      Videos
DataSet                            Downloads   Pictures          yolo_face
evil1323@evil1323:~$ cd darknet/
evil1323@evil1323:~/darknet$
```

執行YOLOv4 (cont.)

- 修改Makfile的內容，如下圖



執行YOLOv4 (cont.)

- 修改Makfile的內容，如下圖

Open ▾ Makefile ~/YoloV3_Demo/darknet

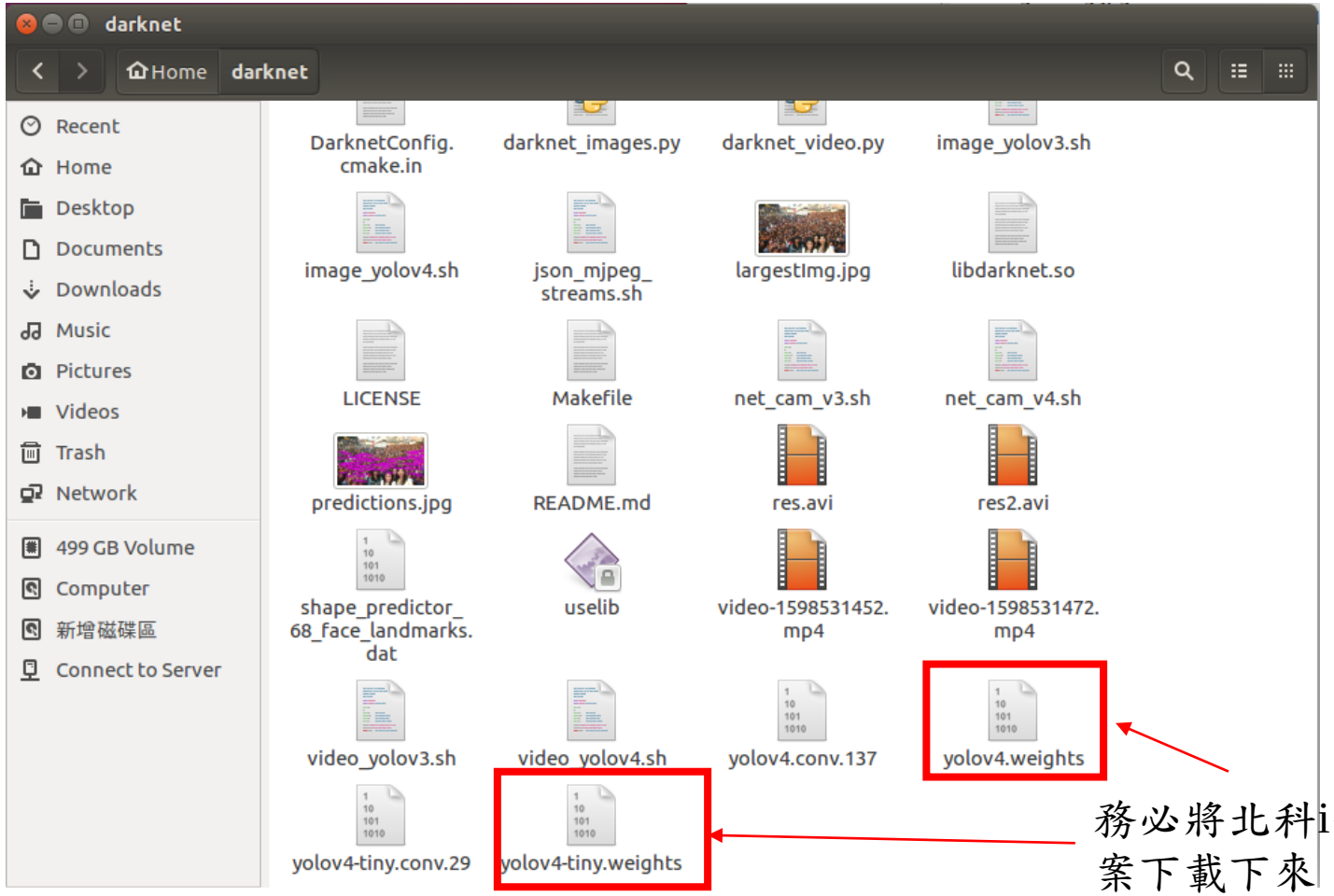
```
# ARCH= -gencode arch=compute_53,code=[sm_53,compute_53]

# For Jetson Tx2 or Drive-PX2 uncomment:
# ARCH= -gencode arch=compute_62,code=[sm_62,compute_62]


VPATH=./src/
EXEC=darknet
OBJDIR=./obj/

ifeq ($(LIBSO), 1)
LIBNAMESO=libdarknet.so
APPNAMESO=uselib
endif

ifeq ($(USE_CPP), 1)
CC=g++
else
CC=gcc
endif

CPP=g++ -std=c++11
NVCC=/usr/local/cuda-9.0/bin/nvcc
OPTS=-Ofast
LDFLAGS= -lm -pthread
COMMON= -Iinclude/ -I3rdparty/stb/include
CFLAGS=-Wall -Wfatal-errors -Wno-unused-result -Wno-unknown-pragmas -fPIC
```

執行YOLOv4 (cont.)

- 修改Makfile的內容後
- 輸入指令: sudo make

```
evil1323@evil1323:~/darknet$ sudo make
[sudo] password for evil1323:
Sorry, try again.
[sudo] password for evil1323:
mkdir -p ./obj/
mkdir -p backup
chmod +x *.sh
g++ -std=c++11 -std=c++11 -Iinclude/ -I3rdparty/stb/include -Wall -Wfatal-errors
 -Wno-unused-result -Wno-unknown-pragmas -fPIC -Ofast -c ./src/image_opencv.cpp
-o obj/image_opencv.o
```

執行YOLOv4 (cont.)



務必將北科i學園plus上的權重檔案下載下來，並放置於資料夾內。

Darknet_video.py程式碼說明

darknet_video.py (~/YoloV3_Demo/darknet) - gedit

Open Save

command × darknet_video.py ×

```
from ctypes import *
import math
import random
import os
import cv2
import numpy as np
import time
import darknet

def convertBack(x, y, w, h):
    xmin = int(round(x - (w / 2)))
    xmax = int(round(x + (w / 2)))
    ymin = int(round(y - (h / 2)))
    ymax = int(round(y + (h / 2)))
    return xmin, ymin, xmax, ymax


def cvDrawBoxes(detections, img):
    for detection in detections:
        x, y, w, h = detection[2][0],\
            detection[2][1],\
            detection[2][2],\
            detection[2][3]
        xmin, ymin, xmax, ymax = convertBack(
            float(x), float(y), float(w), float(h))
        pt1 = (xmin, ymin)
        pt2 = (xmax, ymax)
        cv2.rectangle(img, pt1, pt2, (0, 255, 0), 1)
        cv2.putText(img,
                    detection[0].decode() +
                    " [" + str(round(detection[1] * 100, 2)) + "]",
                    (pt1[0], pt1[1] - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
                    [0, 255, 0], 2)
    return img
```

畫出偵測到的物件的Bounding box

putText的參數為:
(影像,添加的文字,左上角坐標,字體,字體大小,顏色,字體粗細)
detection[0].decode()為偵測到的物件類別名稱，可透過這部分來實作出所偵測到的目標物數量。

Darknet_video.py程式碼修改

darknet_video.py (~/YoloV3_Demo/darknet) - gedit

Open Save

```
                pass
        except Exception:
            pass
cap = cv2.VideoCapture(1)
#cap = cv2.VideoCapture("test.mp4")
cap.set(3, 1280)
cap.set(4, 720)
out = cv2.VideoWriter(
    "output.avi", cv2.cv.CV_FOURCC('M', 'J', 'P', 'G'), 10.0,
    (darknet.network_width(netMain), darknet.network_height(netMain)))
print("Starting the YOLO loop...")
```

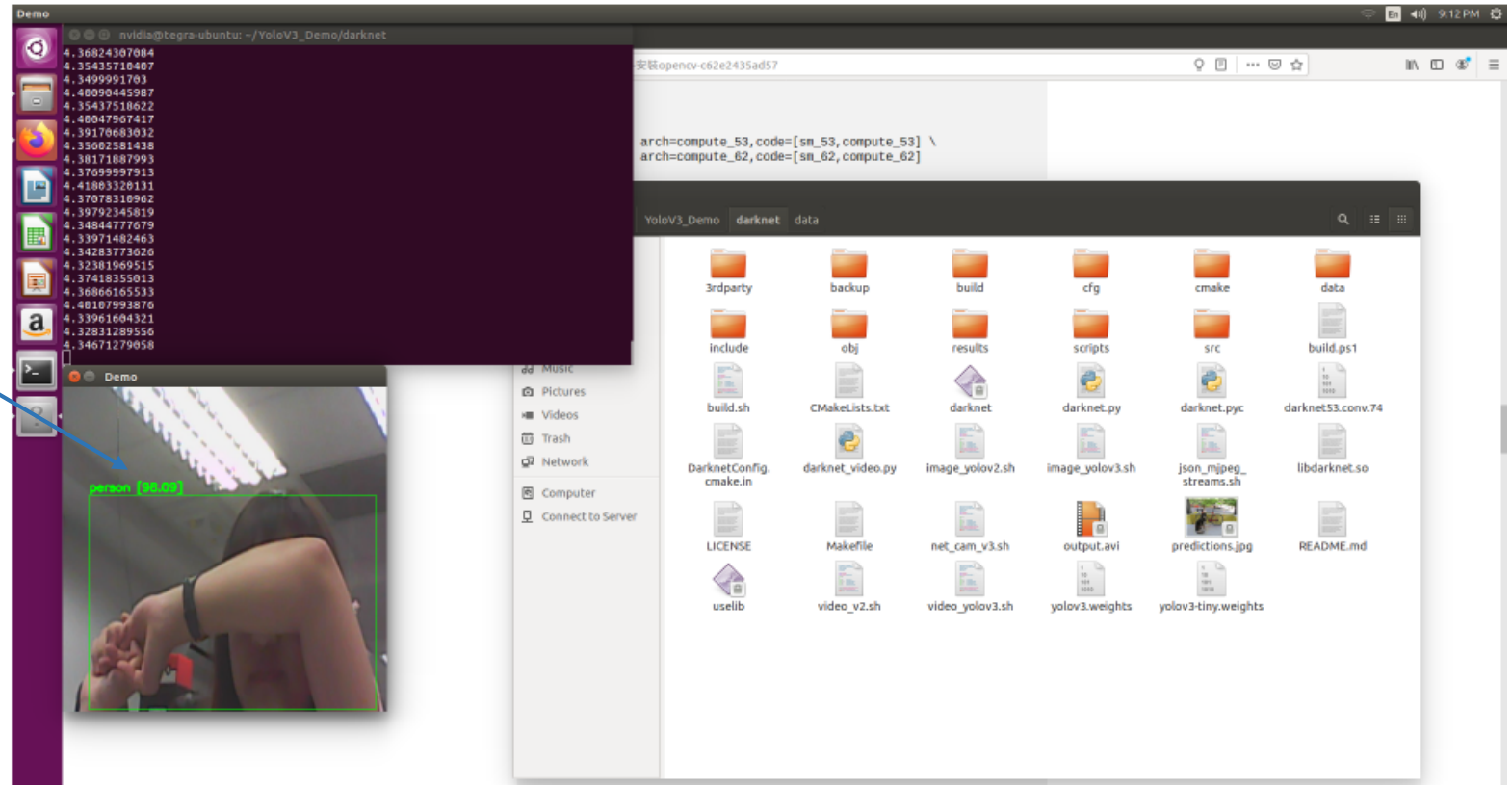
```
cap = cv2.VideoCapture("test.mp4")
cap.set(3, 1280)
cap.set(4, 720)
out = cv2.VideoWriter(
    "output.avi", cv2.VideoWriter_fourcc(*"MJPG"), 10.0,
    (darknet.network_width(netMain), darknet.network_height(netMain)))
print("Starting the YOLO loop...")
```

- 此為opencv2.4.3版本的寫法，若是要使用opencv3寫法請參考黃色框處

使用YOLOv4偵測結果

- 在terminal上執行python darknet_video.py的結果

執行出來的結果為此圖



參考資料

參考資料

- YOLO
  - https://github.com/AlexeyAB/darknet#how-to-train-tiny-yolo-to-detect-your-custom-objects
  - You Only Look Once: Unified, Real-Time Object Detection : https://arxiv.org/pdf/1506.02640.pdf
  - YOLO9000: Better, Faster, Stronger : https://arxiv.org/pdf/1612.08242.pdf
  - YOLOv3: An Incremental Improvement : https://arxiv.org/pdf/1804.02767.pdf
  - YOLOv4: Optimal Speed and Accuracy of Object Detection: https://arxiv.org/pdf/2004.10934.pdf