物件辨識

Object Detection

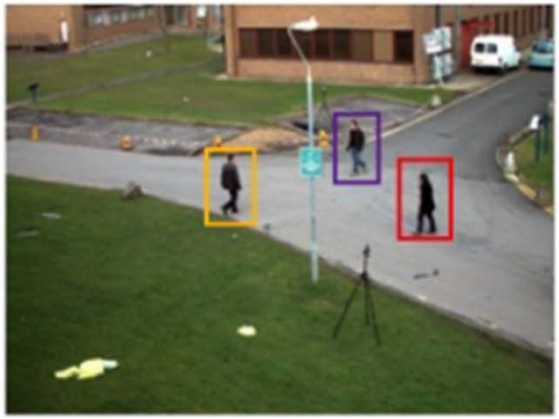
Outline

- Introduction
- Object Detection
  - Method
    - Sliding Window
    - Region Proposal: Selective Search
  - YOLO
  - RCNN Series
- Tracker
  - Centroid Tracking
  - SORT
  - Deep SORT

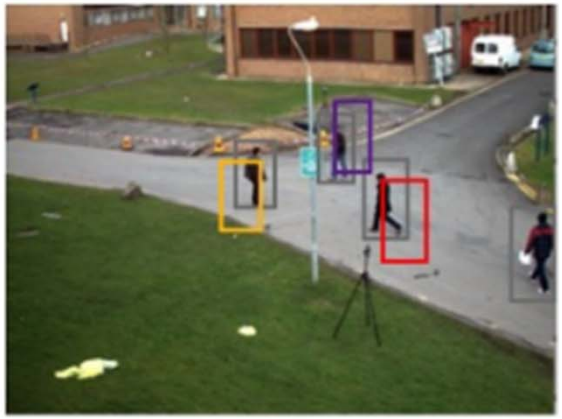
影片 SORT(Simple Online Realtime Tracking)
https://www.youtube.com/watch?v=MYbjjg_Mics

Object Tracking

- Object tracking consists of two parts:
  - **Object Detection**
    - A deep learning method for locating objects in images or videos.
  - **Tracker**
    - Determine whether the objects captured by the frame before and after belong to the same one or not. As shown below, the 3 objects (yellow, purple, red) are detected by frame t, and the new objects (gray box) are detected by frame t+1.



Frame t



Frame t+1

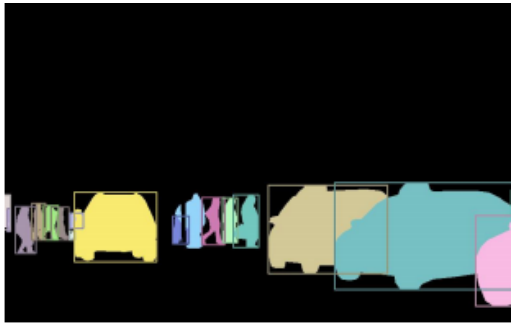
What is Object Detection?

Deep Learning for Computer Vision Tasks

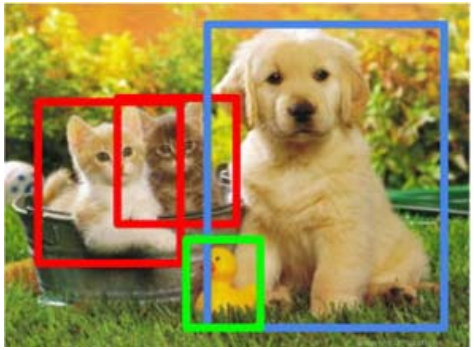
CAT

Image Classification

Image

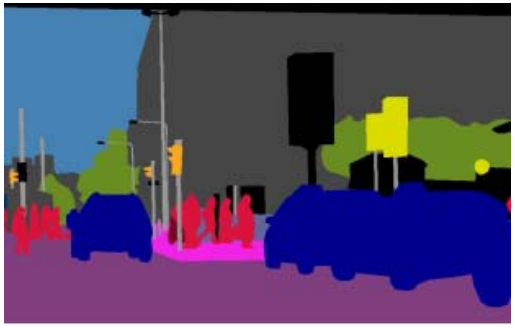


Semantic Segmentation

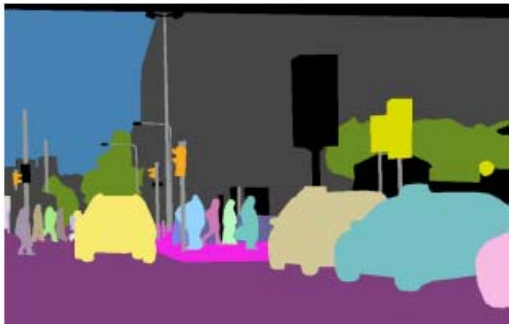


Cat
Dog
Duck

Object Detection

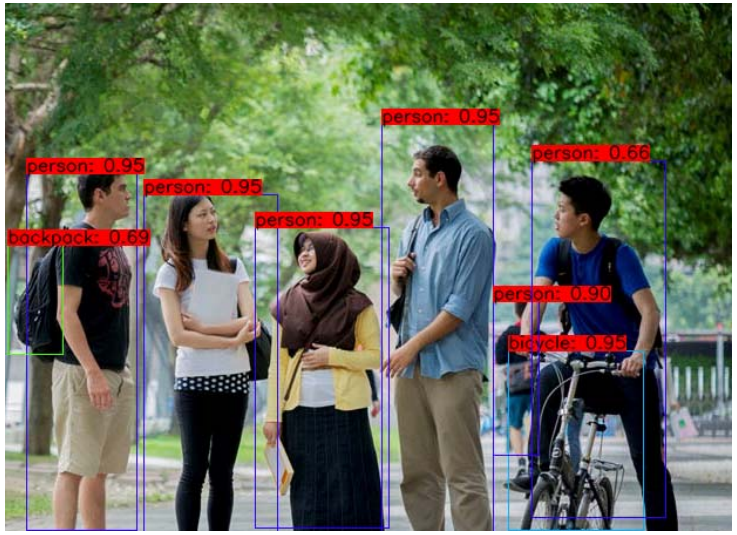


Instance Segmentation



Panoptic Segmentation
全景

Object Detection



Give an image

Detect the objects with the specified categories in the image.

Introduction

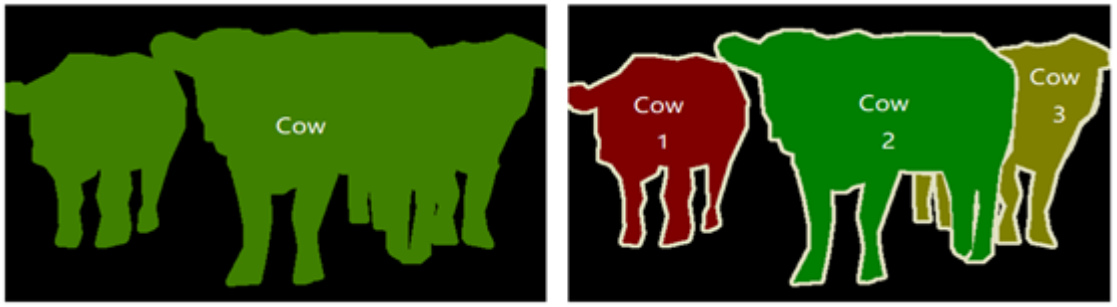
- In the field of computer vision, there are several fundamental visual recognition problems:
  - **Image classification**, aims to recognize semantic categories of objects
  - **Object detection + localization** not only recognizes object categories, but also predicts the location of each object by a bounding box

(a) Image Classification (b) Object Detection

Semantic / Instance Segmentation

- **Semantic segmentation (SS)**,
  - Aim to predict pixel-wise classifiers to assign a specific category label to each pixel, thus providing an even richer understanding of an image.
  - Semantic segmentation does not distinguish between multiple objects of the **same category**.
- **Instance segmentation (IS)**
  - A relatively new setting at the **intersection of object detection and semantic segmentation.**
  - To identify different objects and assign each of them a separate categorical **pixel-level mask**.
  - A special setting of object detection, where instead of localizing an object by a bounding box, **pixel-level localization** is desired.

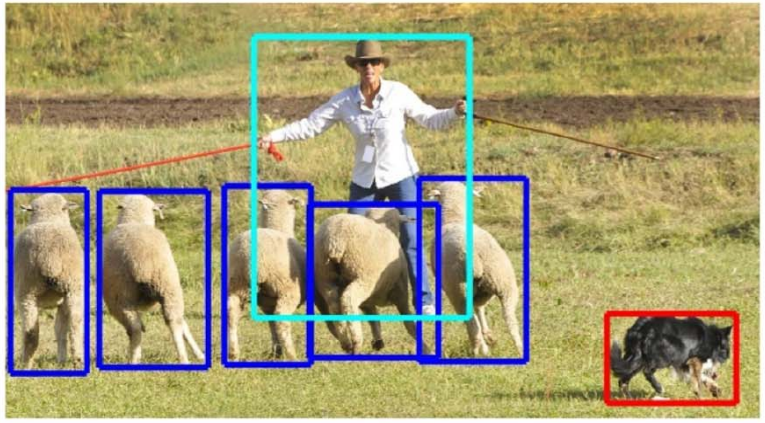
SS



IS

Instance Segmentation

(a) Image classification



(b) Object localization

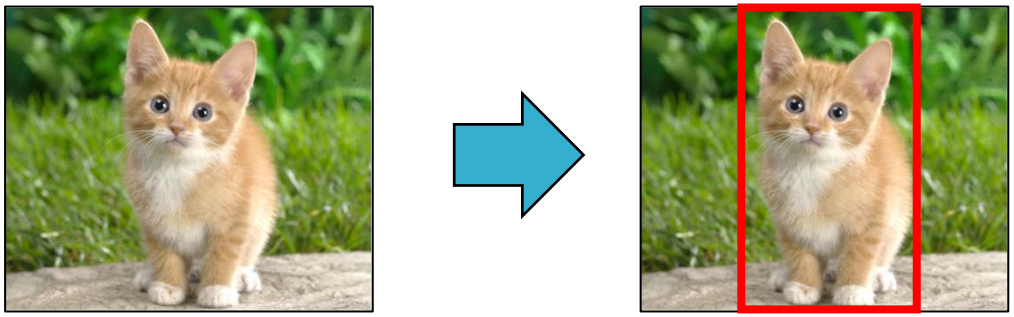
(c) Semantic segmentation

(d) Instance Segmentation

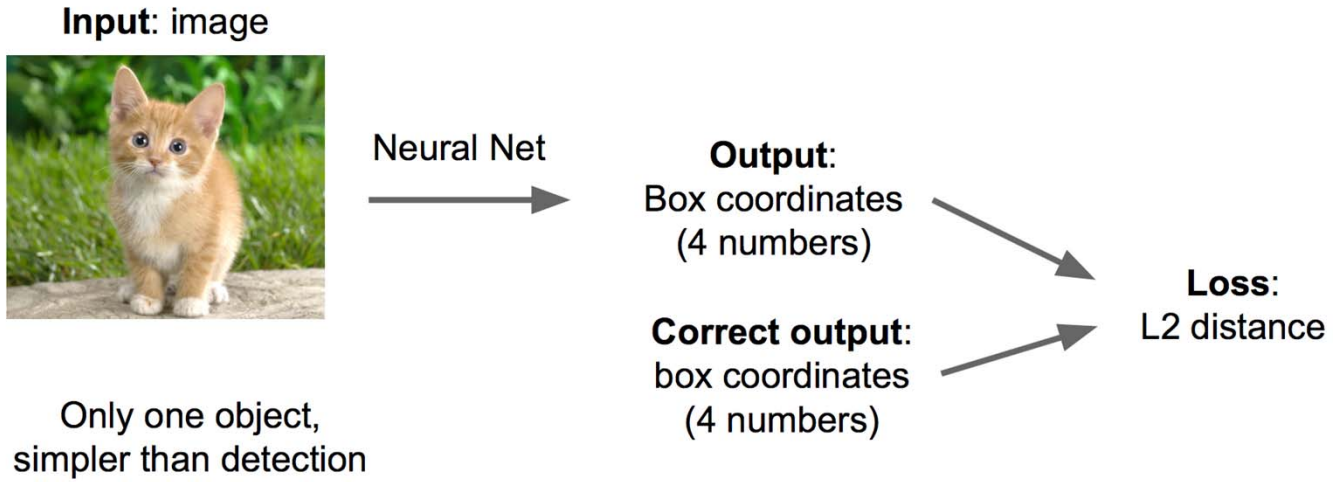
Microsoft COCO: Common Objects in Context

Objection Localization

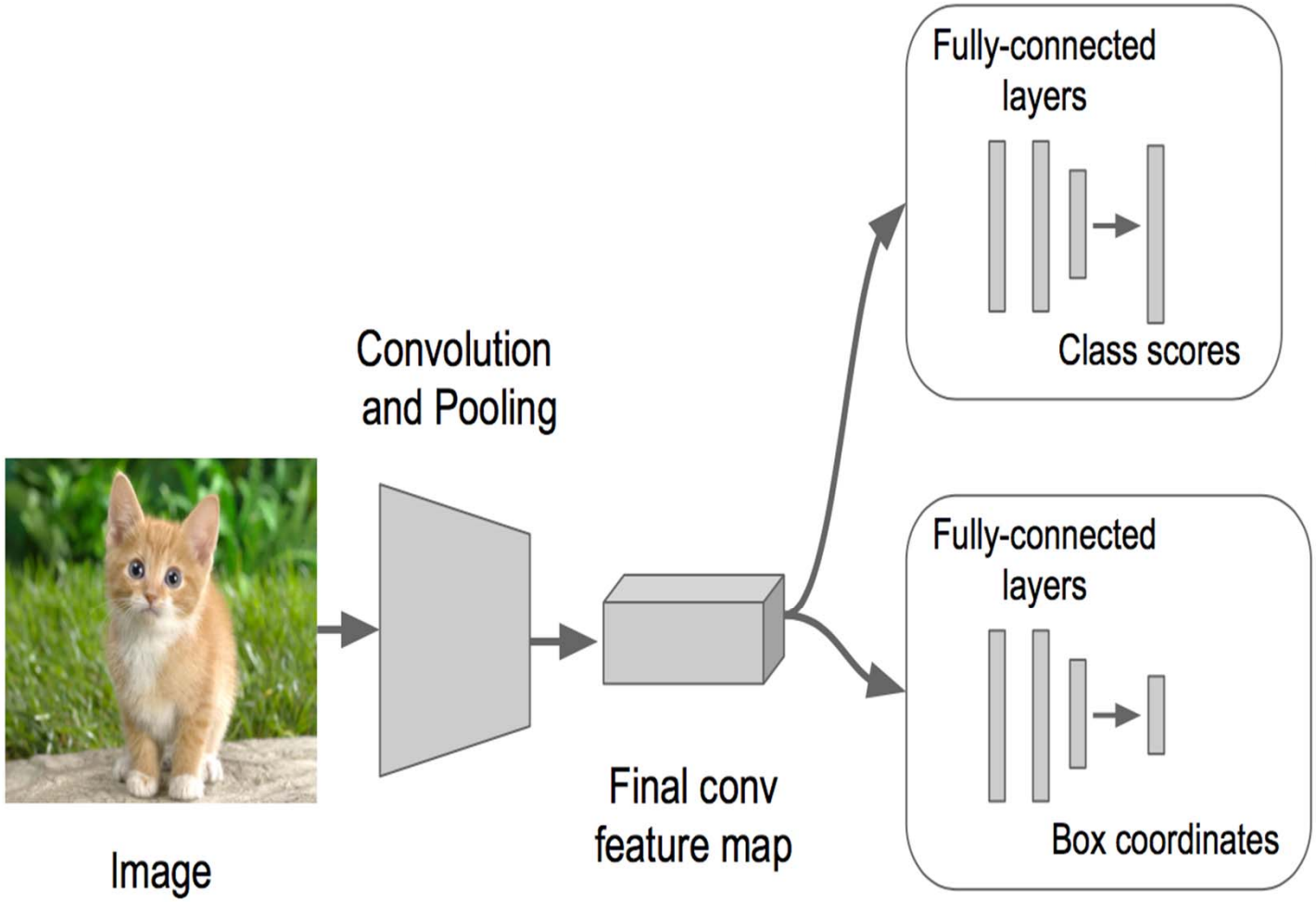
- Suppose an image contains a single object. How might we predict the object's bounding box? i.e. localize it.



- One possible setup: regression



Object Localization



Classification Head:

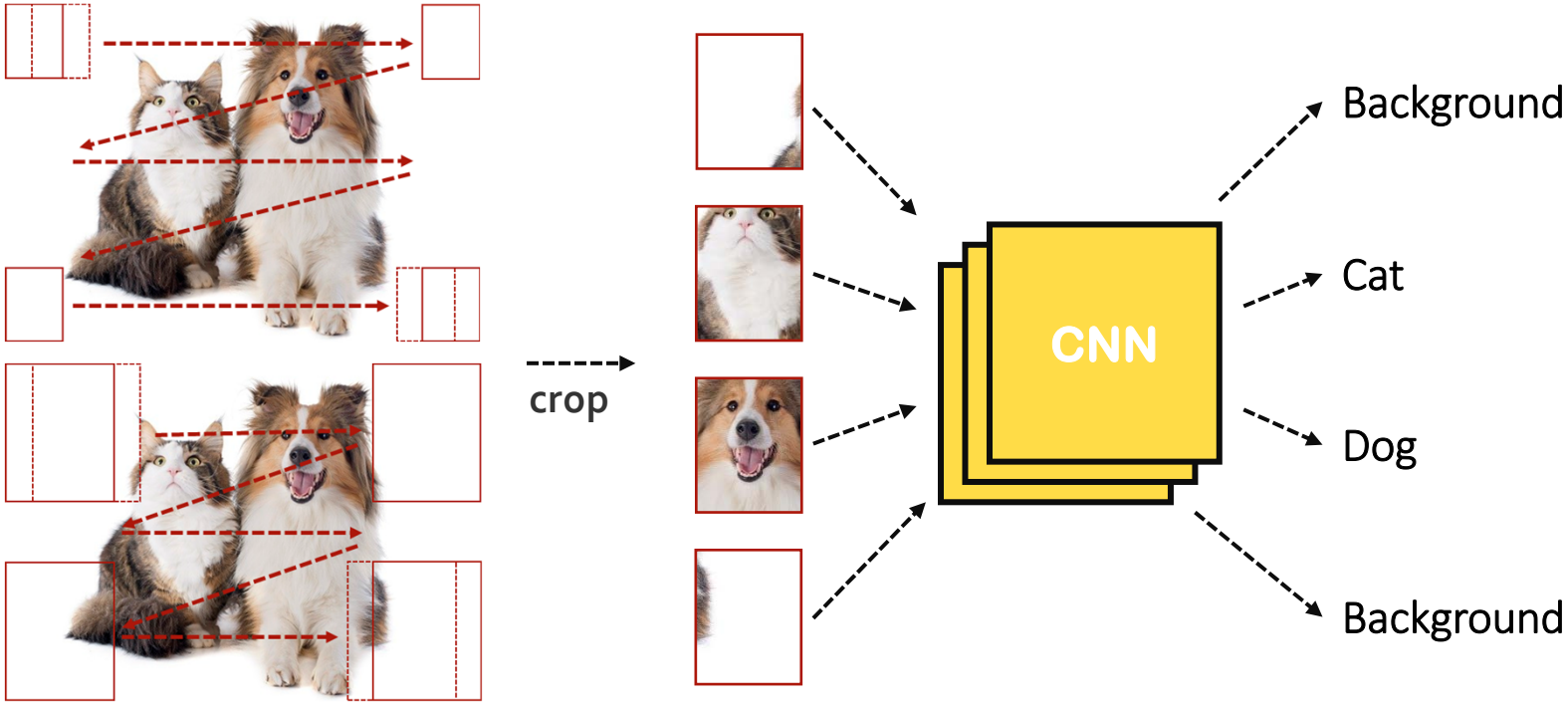
- C Scores for C classes

Localization Head:

- Class agnostic: (x,y,w,h)
- Class specific: (x,y,w,h) X C

Object Proposal

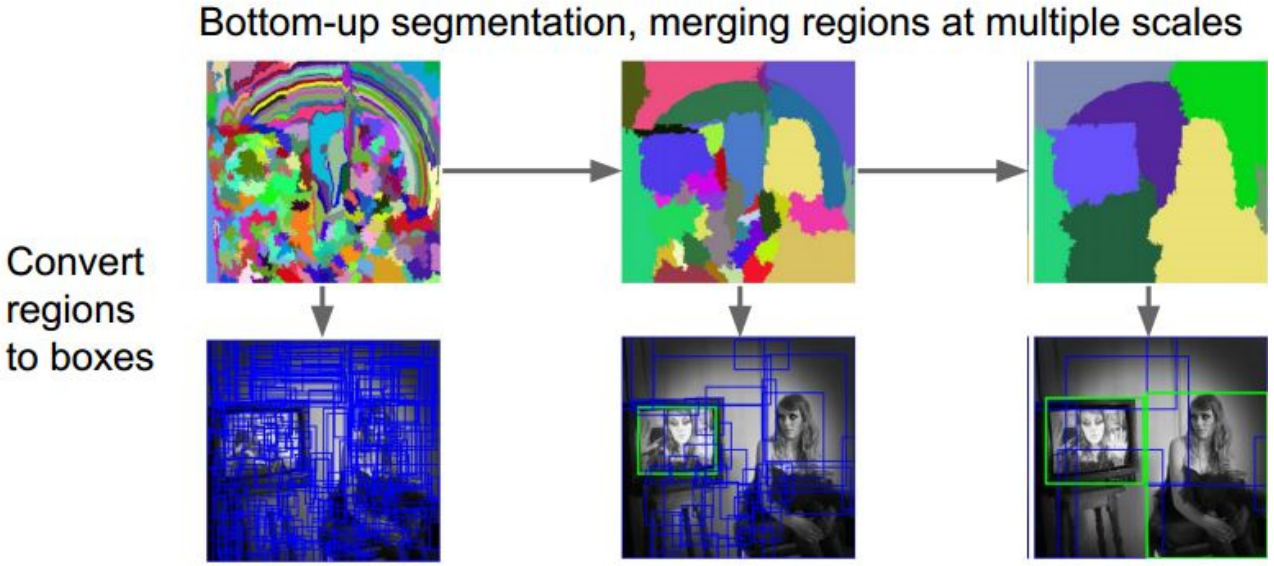
- **Sliding Window**
  - Apply a CNN model to the cropped regions in the image, classifying each region as an object or background.



Problem: Need to apply the CNN model to a huge number of locations and scales. Very computationally expensive!

Region Proposal

- **Selective Search**
  - Add all bounding boxes corresponding to segmented parts to the list of the **regional proposals**.
  - Group adjacent segments based on similarity.
  - About 2000 bounding boxes need to be fed into CNN.



Uijlings et al, "Selective Search for Object Recognition", IJCV 2013

Object Detection

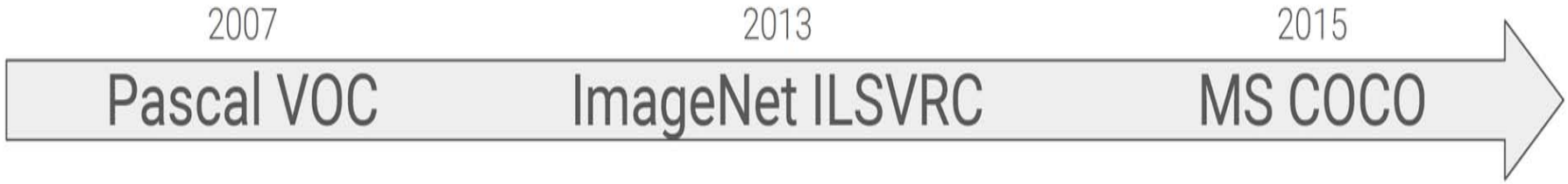
- In fact, object detection is the basic step towards many computer vision applications, such as
  - face recognition,
  - Pedestrian detection,
  - video analysis,
  - logo detection.

Deep Learning-based Methods

Deep Learning-based methods

- The deep learning-based methods (2010-2020)
  - In **2012**, with the big image data (i.e., **ImageNet**), deep CNN network (called **AlexNet**) achieves the best detection performance on ImageNet ILSVRC2012, which outperforms the second best method by **10.9%** on ILSVRC-2012 competition.
  - The **mAP** of object detection on PASCAL VOC2007 dramatically increases from 58% (based on RCNN with AlexNet) to 86% (based on Faster RCNN with ResNet).
  - Currently, the state-of-the-art methods for deep object detection are based on deep convolutional neural networks (CNN)

Object Detection: Datasets



- 20 Classes
- 11K Training images
- 27K Training objects

Was de-facto standard, currently used as quick benchmark to evaluate new detection algorithms.

- 200 Classes
- 476K Training images
- 534K Training objects

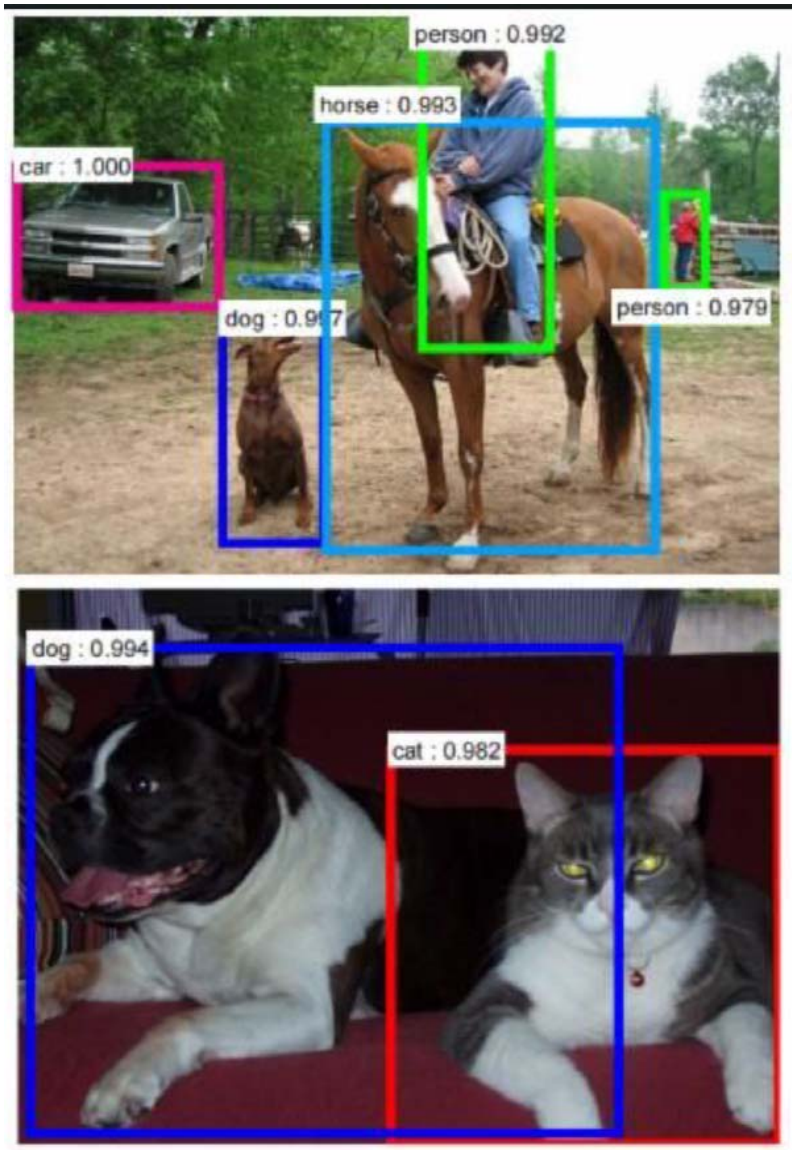
Essentially scaled up version of PASCAL VOC, similar object statistics.

- 80 Classes
- 200K Training images
- 1.5M Training objects

More categories and more object instances in every image. Only 10% of images contain a single object category, 60% in Pascal. More small objects than large objects.

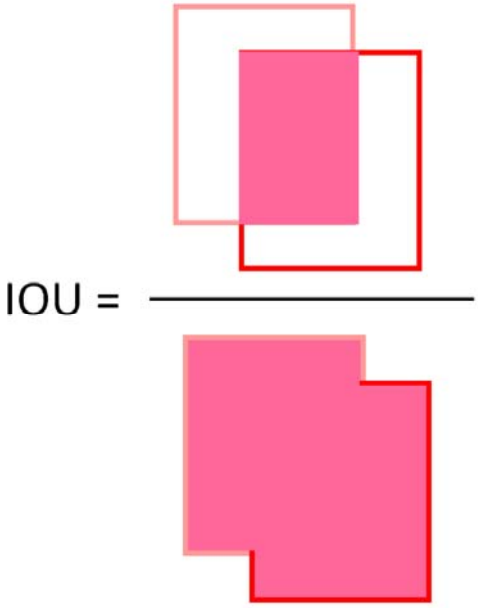
Output of Object Detection

- Input:
  - Image
- Output:
  - For each object class **c** and each image **i**, an algorithm returns **predicted detection**:
    - class,
    - locations and
    - confidence scores.



Evaluation metrics

- **IOU (Intersection over Union)**
  - 比值：即兩個 bndBox 的交集 / 兩個 bndBox 的聯集，也就是指 **predict** 的 bndBox與 **Ground Truth** 的 bndBox 的交集除以聯集，通常score > 0.5 就被認為是不錯的結果了
- 在 object detection中，會將預測目標物件與 Ground Truth 做 IOU計算，若 IOU 大於**閥值** (Threshold，通常設定0.5)，。

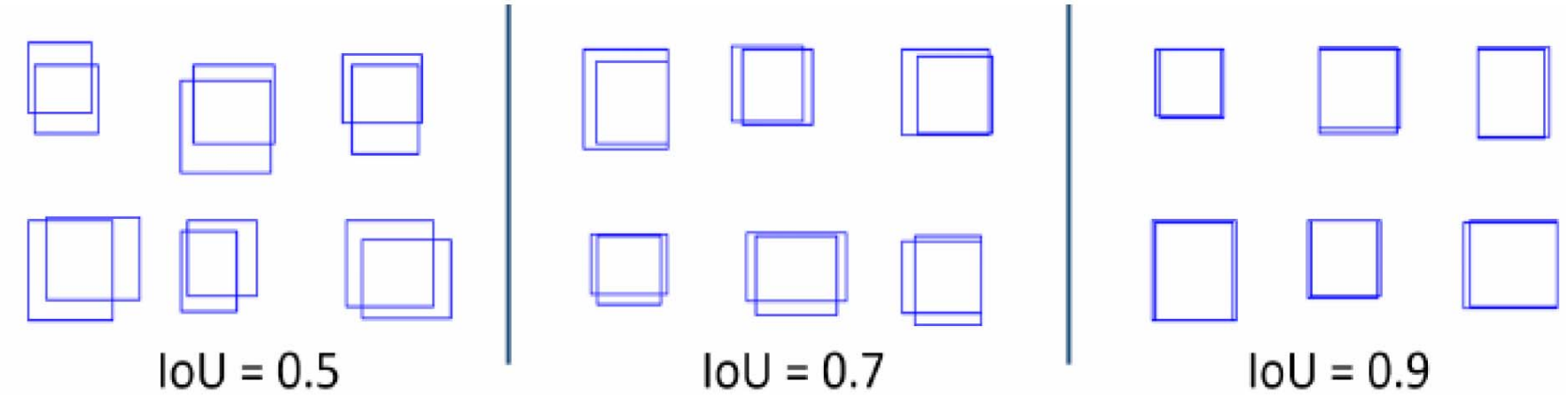


IOU

https://medium.com/@chingi071/yolo-c49f70241aa7

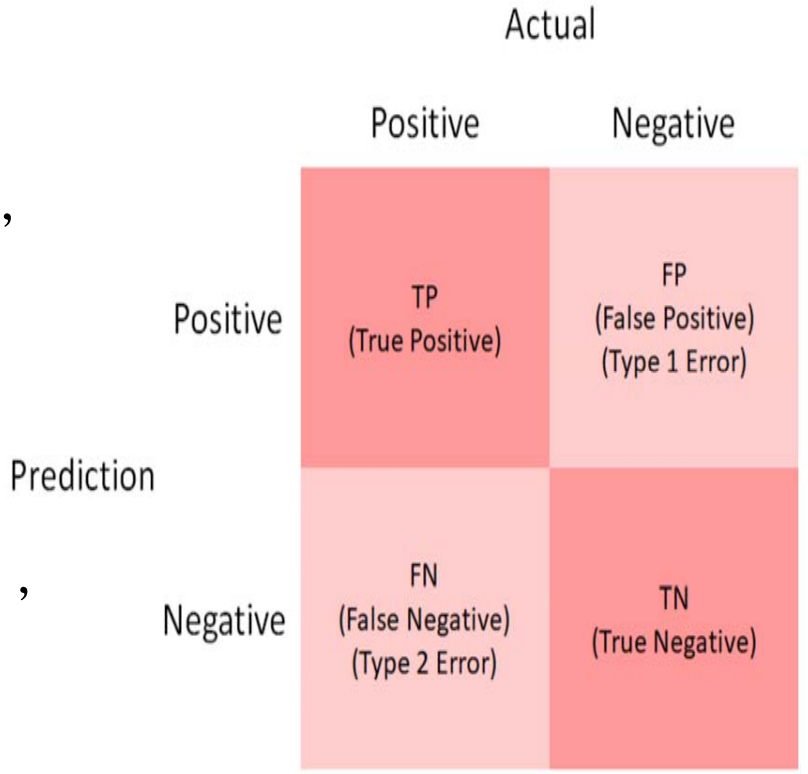
Quality Assessment and Metrics

A few example of IoU values and their associated configuration



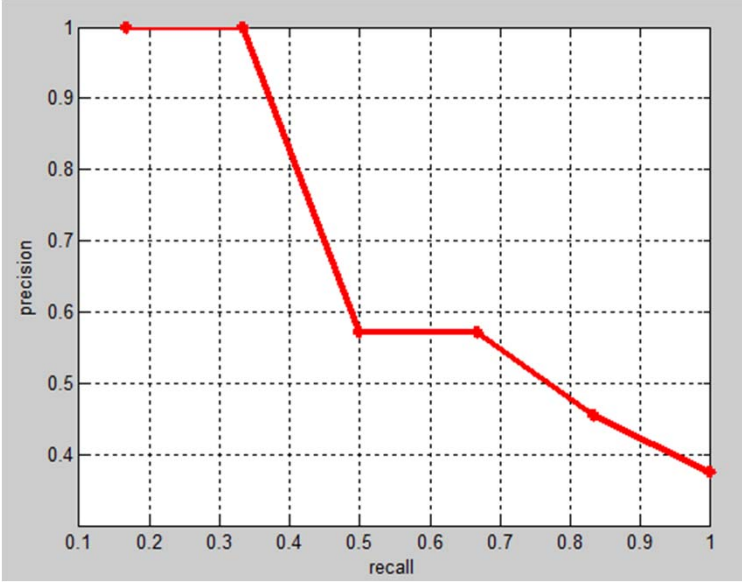
Evaluation metric

- **Precision (準確率): TP / (TP + FP)**
- **Recall (召回率): TP / (TP + FN)**
- Confusion Matrix 的四個指標: TP, TN, FP, FN
  - **TP (True Positive)**: 實際為目標物件，也正確地預測出是目標物件
  - **TN (True Negative)**: 實際不為目標物件，也正確地預測出不是目標物件
  - **FP (False Positive)**: 實際不為目標物件，但卻錯誤地預測成是目標物件，也稱作 Type 1 Error
  - **FN (False Negative)**: 實際為目標物件，但卻錯誤地預測成不是目標物件 (或是指沒預測出來的正樣本)，也稱作 Type 2 Error



mAP (mean Average Precision)

- mAP稱做11-point interpolated average precision:
- 首先針對算法畫出PR曲線，然後根據公式計算出AP，
- 意思就是說找出**recall大於某一個閥值(ex:0.3)** 的最大準確率，總共有11個要找，閥值分別為0、0.1、…1，全部找到後取平均就是每一類的AP，而mAP即是對所有分類都做一次求AP的動作然後再取平均。

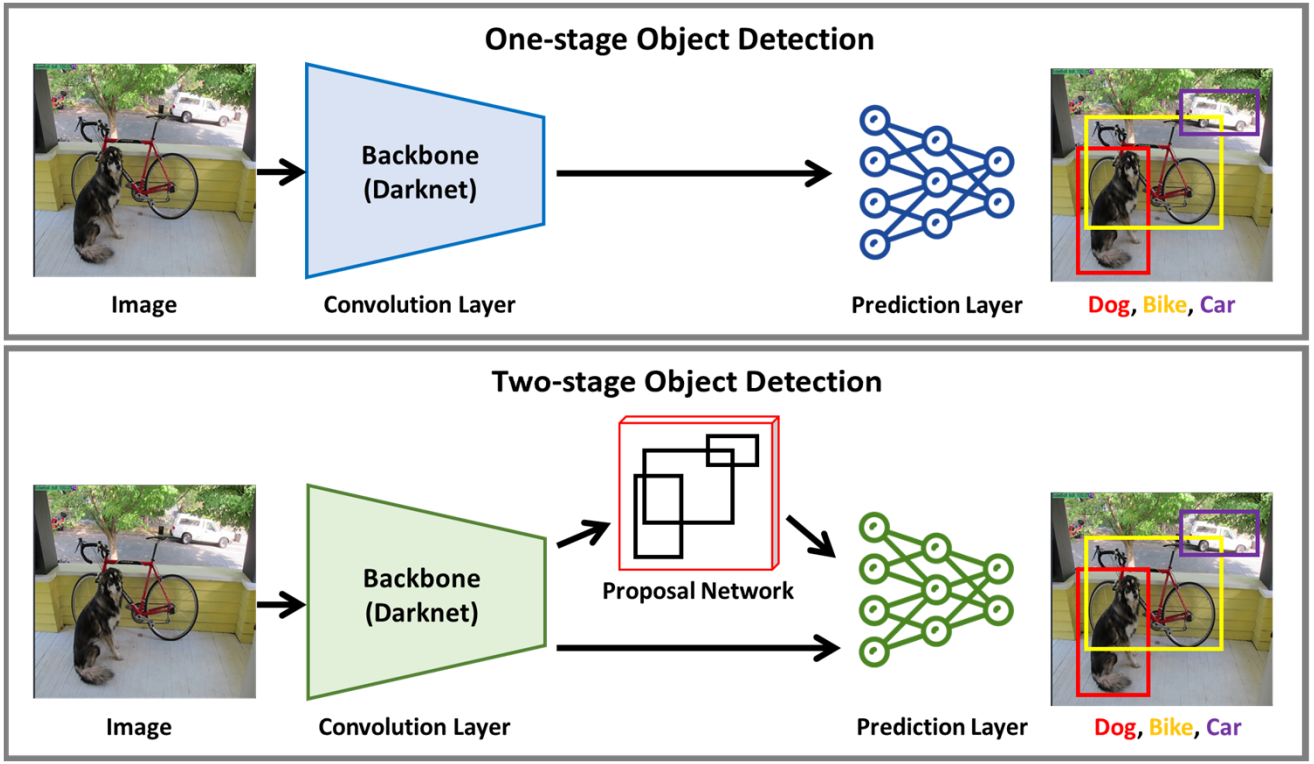


https://darren1231.pixnet.net/blog/post/347752732-machine-learning-map%28mean-average-precision%29-%E8%A9%B3%E7%B4%B0%E8%A7%A3%E8%AA%AA

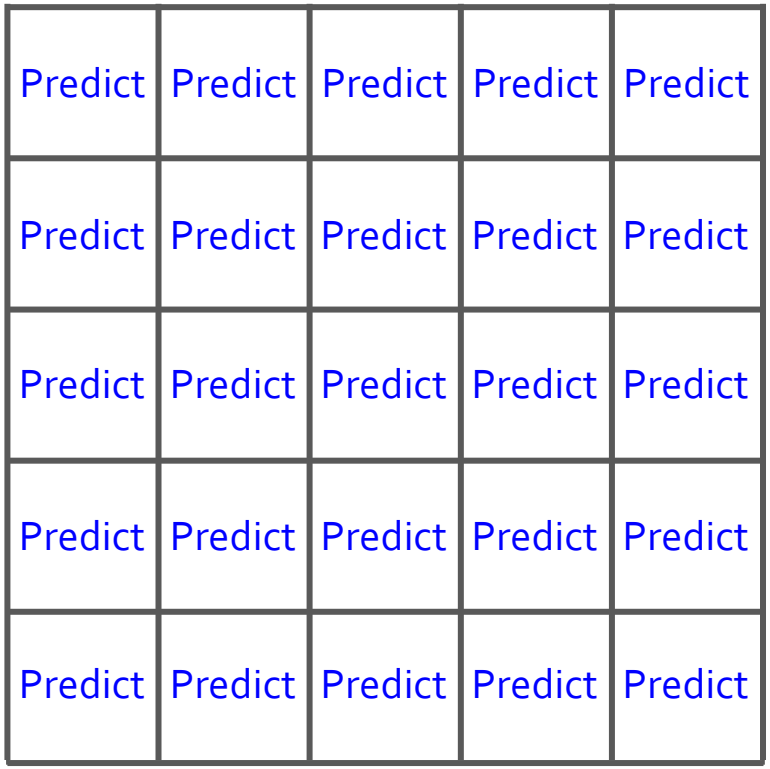
One-stage object detector
vs.
Two stage object detector

What are One-stage & Two-stage?

- There are two different approaches to object detection
  - **One-stage**: Predict the bounding boxes and classes directly.
  - **Two-stage**: Leverage a proposal network to find objects, then use a second network to fine-tune these proposals and output a final prediction.

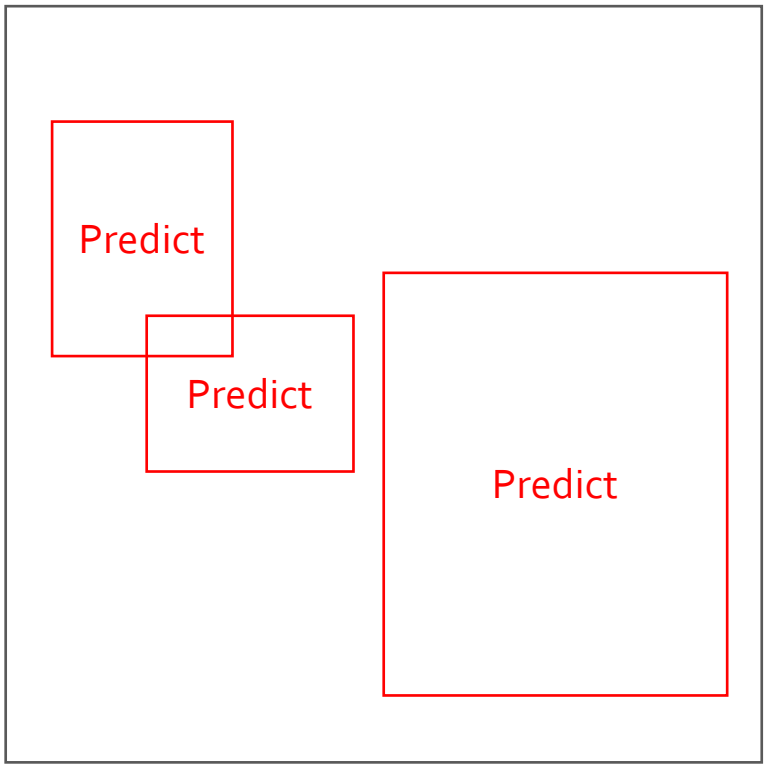


What are One-stage & Two-stage?



Dense prediction

(One-stage)

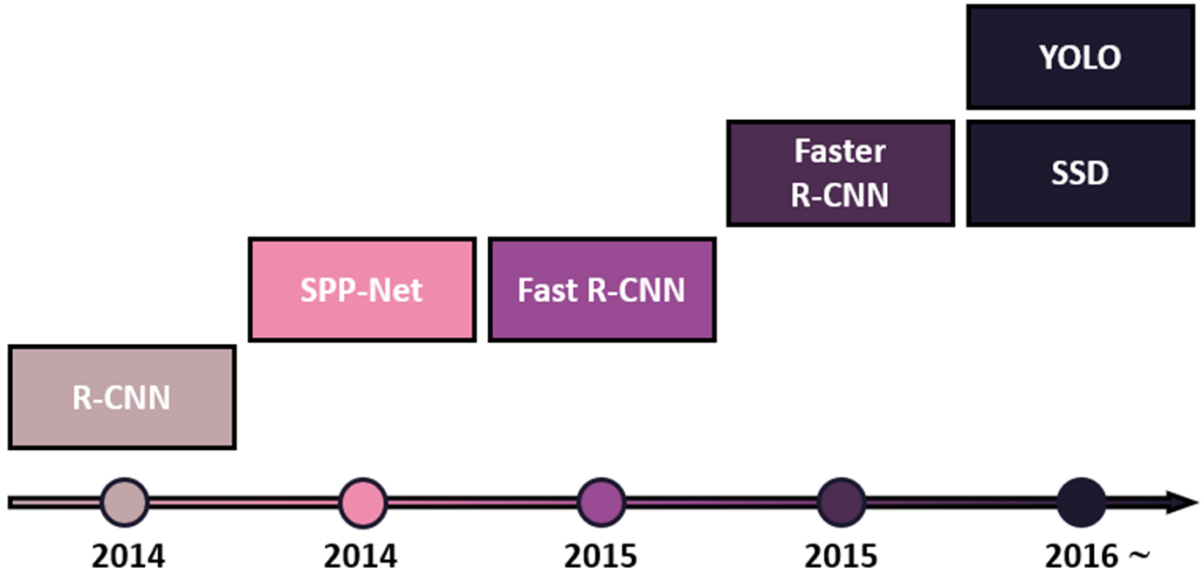


Sparse prediction

(Two-stage)

Well-known Architectures

- Well-known methods of one-stage object detection:
  - YOLO (You Only Look Once) series: YOLO, YOLOv2, YOLOv3, and Tiny YOLO, YOLOV4,~YOLOV7
- Well-known methods of two-stage object detection:
  - R-CNN (Regions with CNN features), Fast R-CNN, and Faster R-CNN
  - SPP-Net



The Deep Learning architecture of Object Detection

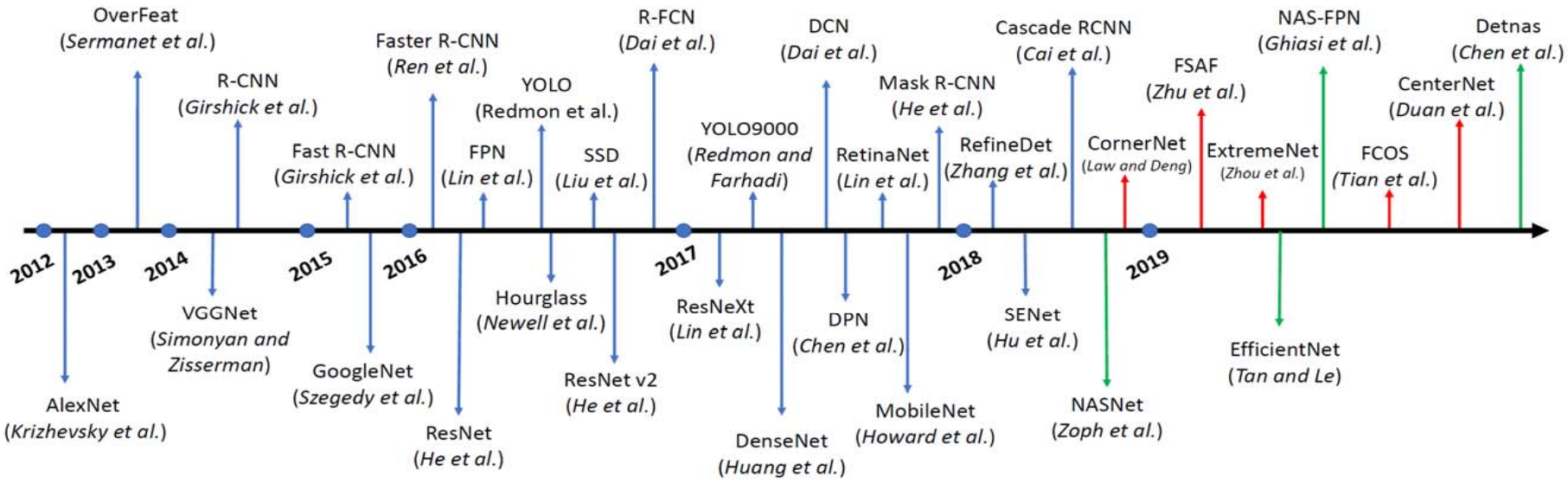
- The methods can be divided into two main classes
  - **Two-stage methods**
    - Firstly generate some **candidate object proposals** and then **classify** these proposals into the specific categories.
    - It has a relatively slower detection speed and higher detection accuracy.

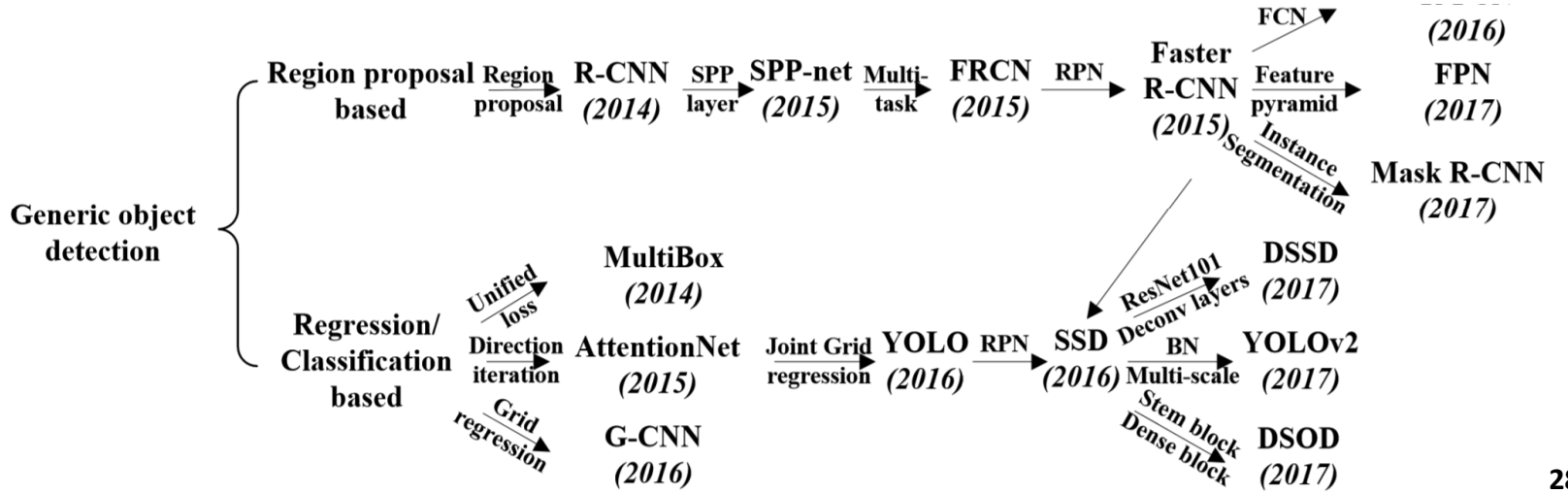
R-CNN
SPPnet
Fast RCNN
Faster RCNN
R-FCN
…

  - **One-stage methods**
    - Simultaneously extract and classify all the object proposals.
    - Have a much faster detection speed and comparable detection accuracy.

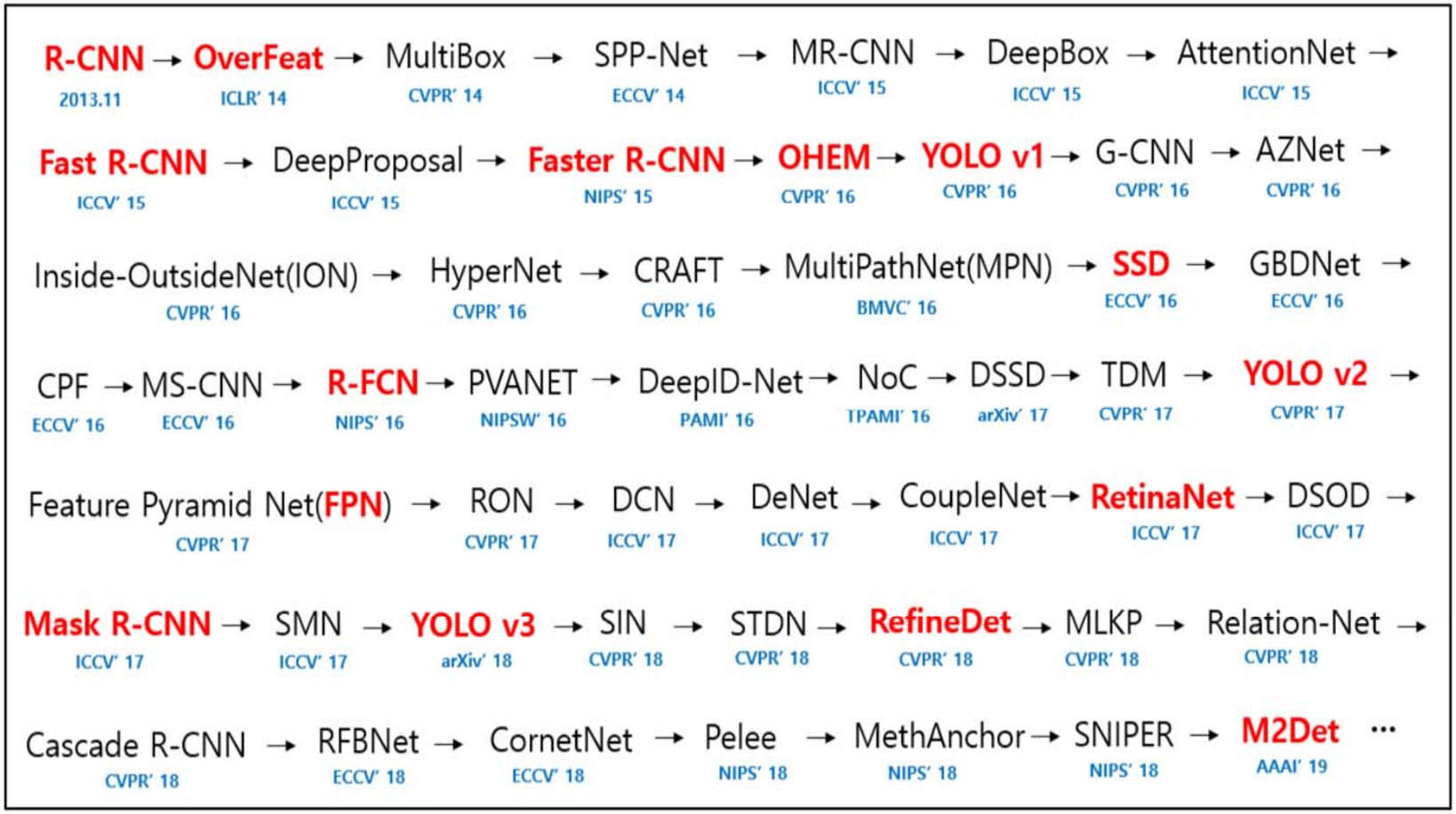
Overfeat
YOLO
SSD
RetinaNet
…

Major milestone in object detection research since 2012





Object-detection with DL



https://github.com/hoya012/deep_learning_object_detection

YOLO

You Only Look Once

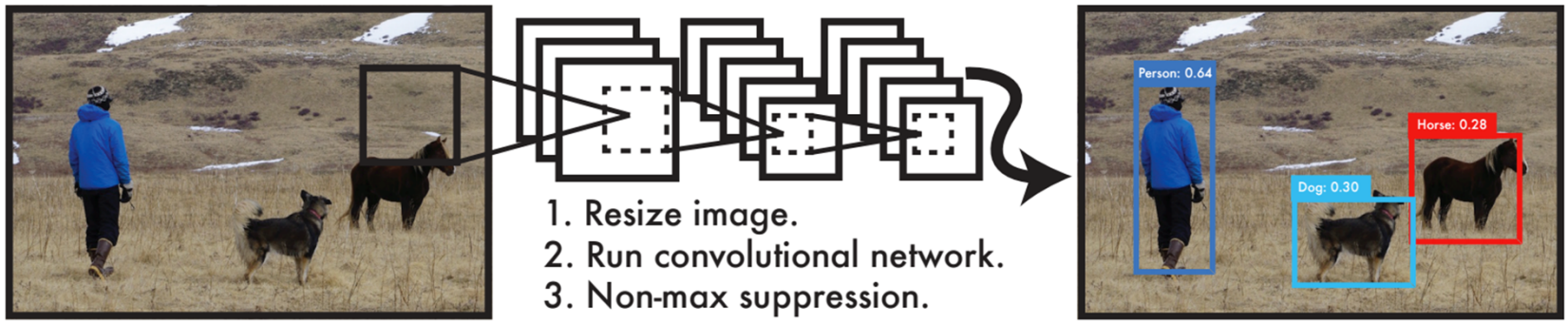
YOLO (You Only Look Once): Unified, Real-Time Object Detection

- A single neural network, unified architecture, and use **Darknet** framework
- Real-time object detection using a simple architecture
  - 10s to 100s of frames/sec

- Solution and Advantages:
  - Simpler structure of network
  - Much more faster, even with real-time property: 150 fps: able to process streaming video in real-time with less than 25 milliseconds of latency
  - Maintaining a proper accuracy range

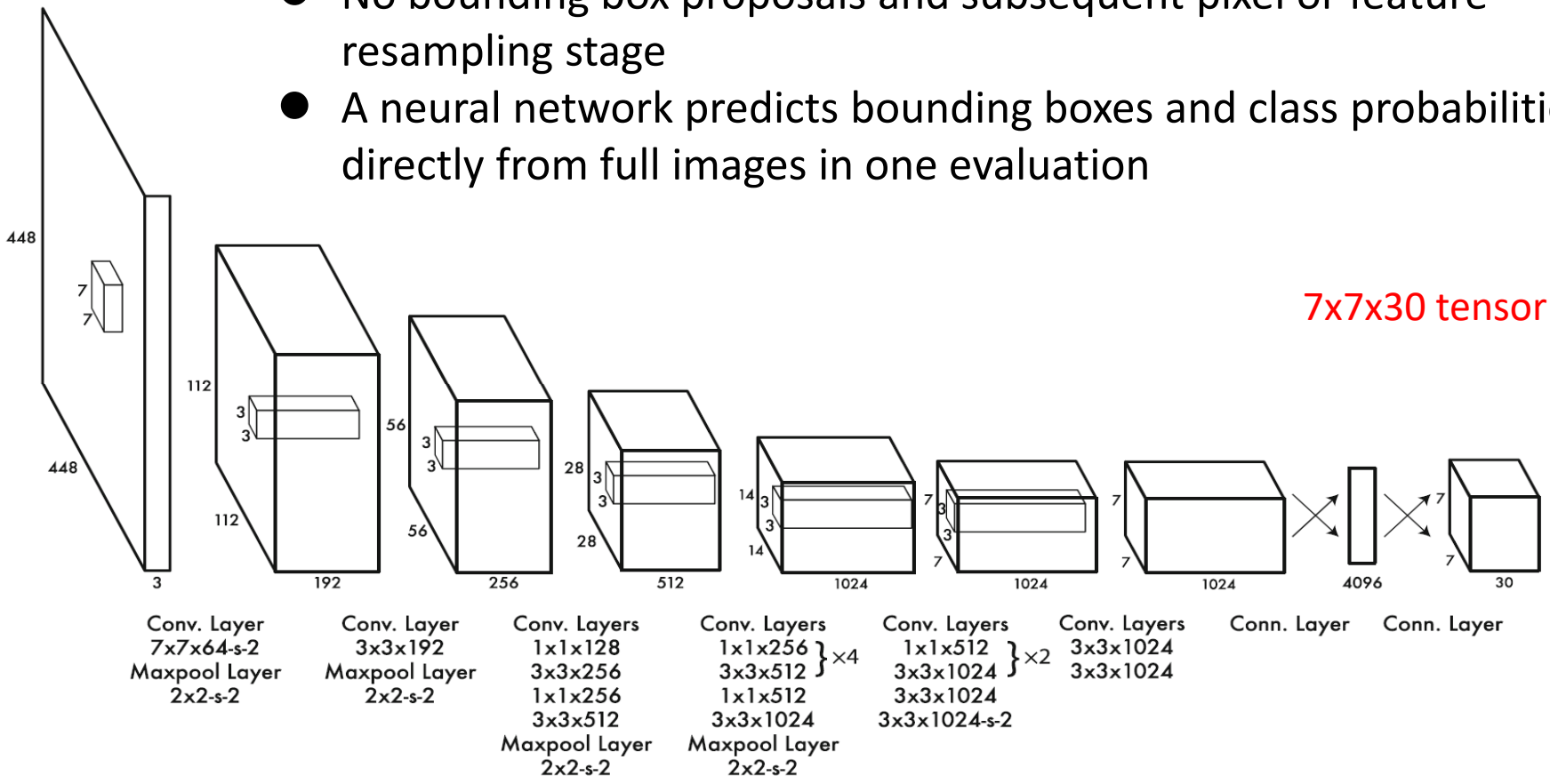
YOLO

- The input image size of YOLO is fixed (i.e., **448×448**).
- YOLO divides the input image into the $k \times k$ grids (k=7).
- Each grid cell predicts *B* bounding boxes with objectness scores and ***C*** conditional class probabilities.
- The predictions of each bounding box are ***(x, y, w, h, s)***, where ***(x, y, w, h)*** gives the location of bounding box and ***s*** is the confidence objectness score of bounding box.
- Thus, the output layer has the size of $k \times k \times (5B + c)$.
- Based on bounding box prediction and corresponding class prediction, YOLO can simultaneously give the object



The architecture of YOLO
24 CNN, 2 FCN

- For speed: Also! structure advantage!
- No bounding box proposals and subsequent pixel or feature resampling stage
- A neural network predicts bounding boxes and class probabilities directly from full images in one evaluation



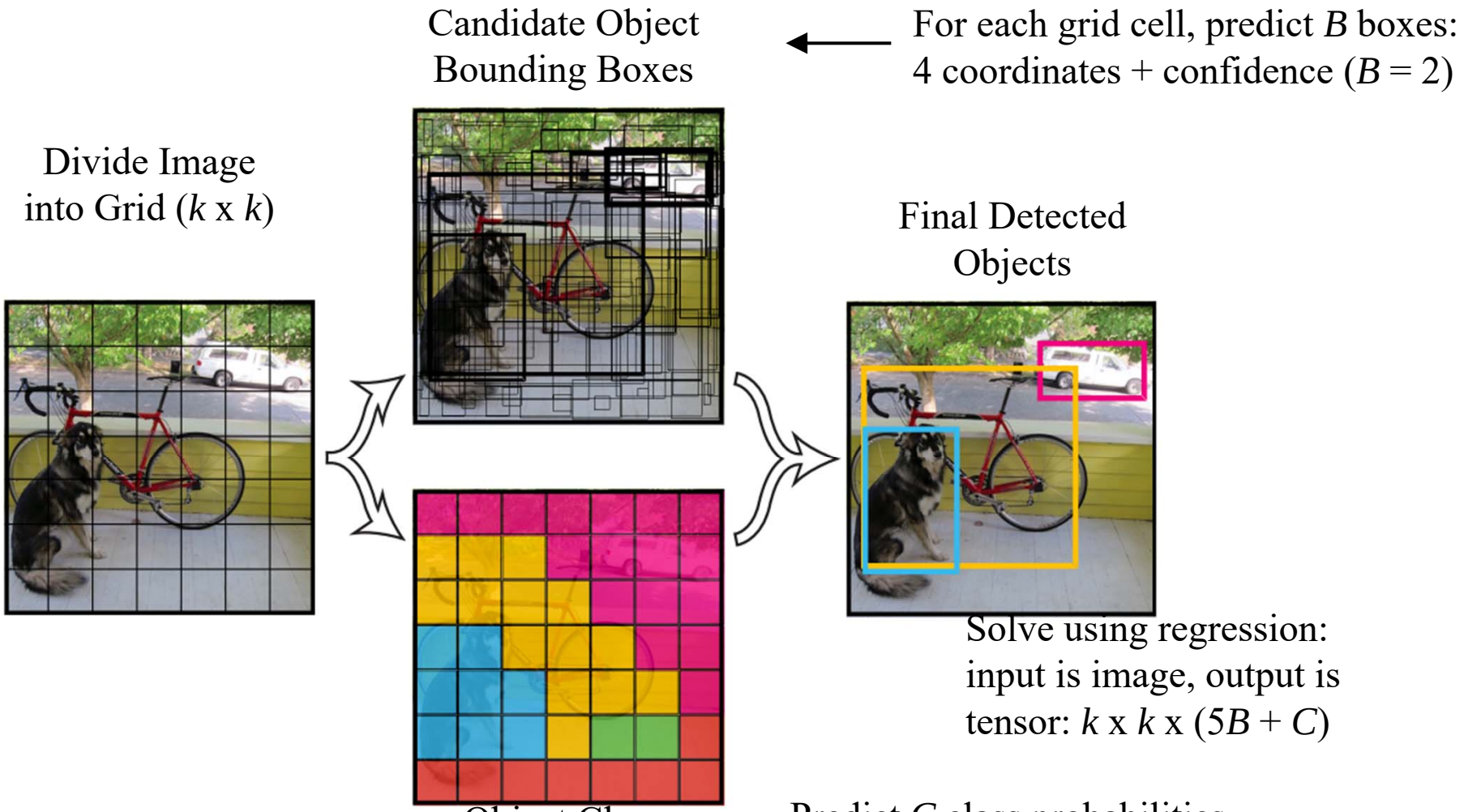
YOLO Details: Boxes and Probabilities

- **Boxes**:
  - (x, y): center of box relative to grid cell
  - (w, h): size relative to whole image

- **Confidence:**
  - If no object in cell, 0
  - If object in cell, IoU between **ground truth** and predicted box
  - **Confidence** = **Pr(Object)*IOU$_{truthpre}$**

- **Class probabilities**:
  - P(Class$_i$ | There is an Object)
  - Multiplied by confidence scores at test time

k=7, B=2 (YOVOv1) PASCAL VOC label C= 20 種，所以 tensor 為 k x k x (5 * B + C) = 7 x 7 x 30



YOLO Versions

- YOLO (darknet) - https://pjreddie.com/darknet/yolov1/ (C++)
- YOLO v2 (darknet) - https://pjreddie.com/darknet/yolov2/ (C++)
  - - Better and faster - 91 fps for 288 x 288
- YOLO v3 (darknet) - https://pjreddie.com/darknet/yolo/ (C++)
- YOLO (caffe) - https://github.com/xingwangsfu/caffe-yolo
- YOLO (tensorflow) - https://github.com/thtrieu/darkflow

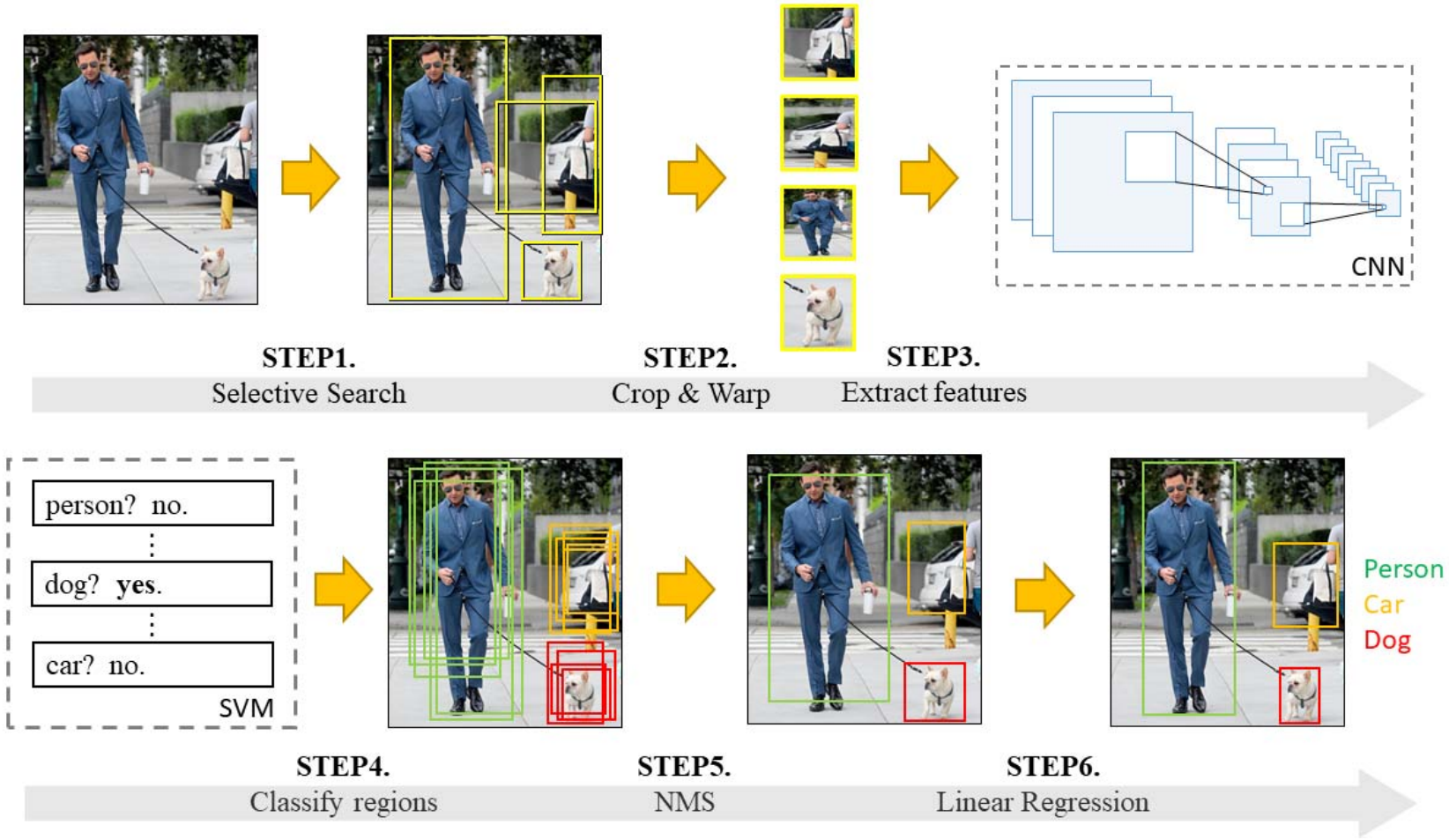
- YOLOv4 https://github.com/AlexeyAB 2020/04
- YOLOv5 https://github.com/ultralytics/yolov5 2020/08
- YOLOv6 https://github.com/meituan/YOLOv6 2022/06
- YOLOv7 https://github.com/WongKinYiu/yolov7 2022/07

R-CNN

- Process
  - Use **Selective Search** to extract around 2000 region proposals.
  - Warp each proposal and propagate it through the CNN model to compute the features.
  - SVM (Support Vector Machine) is used to do classification.
  - Non-maximum Suppression (NMS) will filter the bounding boxes.
  - The bounding box position will be adjusted by linear regression.

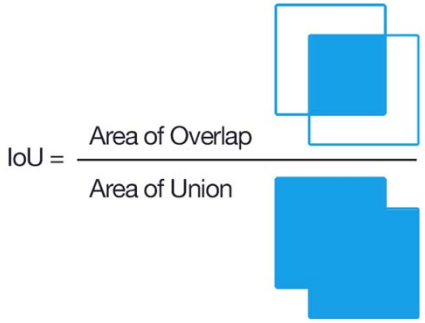
R-CNN

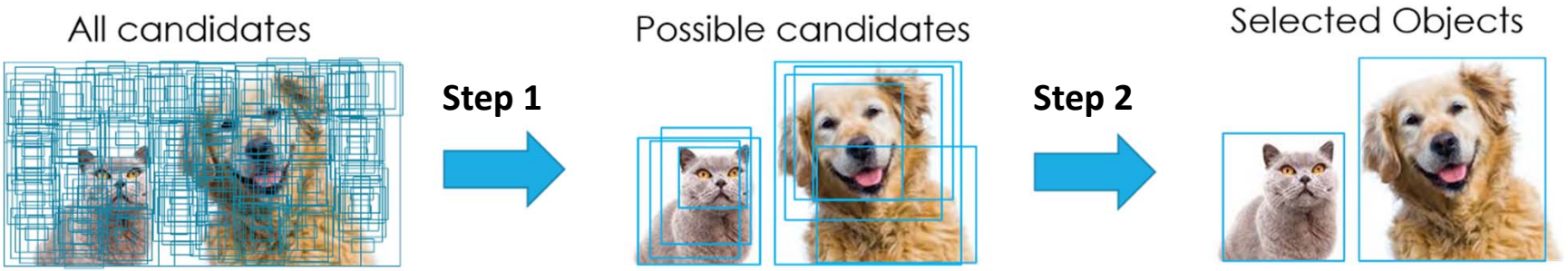
- Process



Non-maximum Suppression (NMS)

- **Step 1:** Select the bounding boxes if their confidence scores are higher than the threshold.
- **Step 2:** Calculate the **IoU** of each bounding box to remove the overlapping bounding boxes.



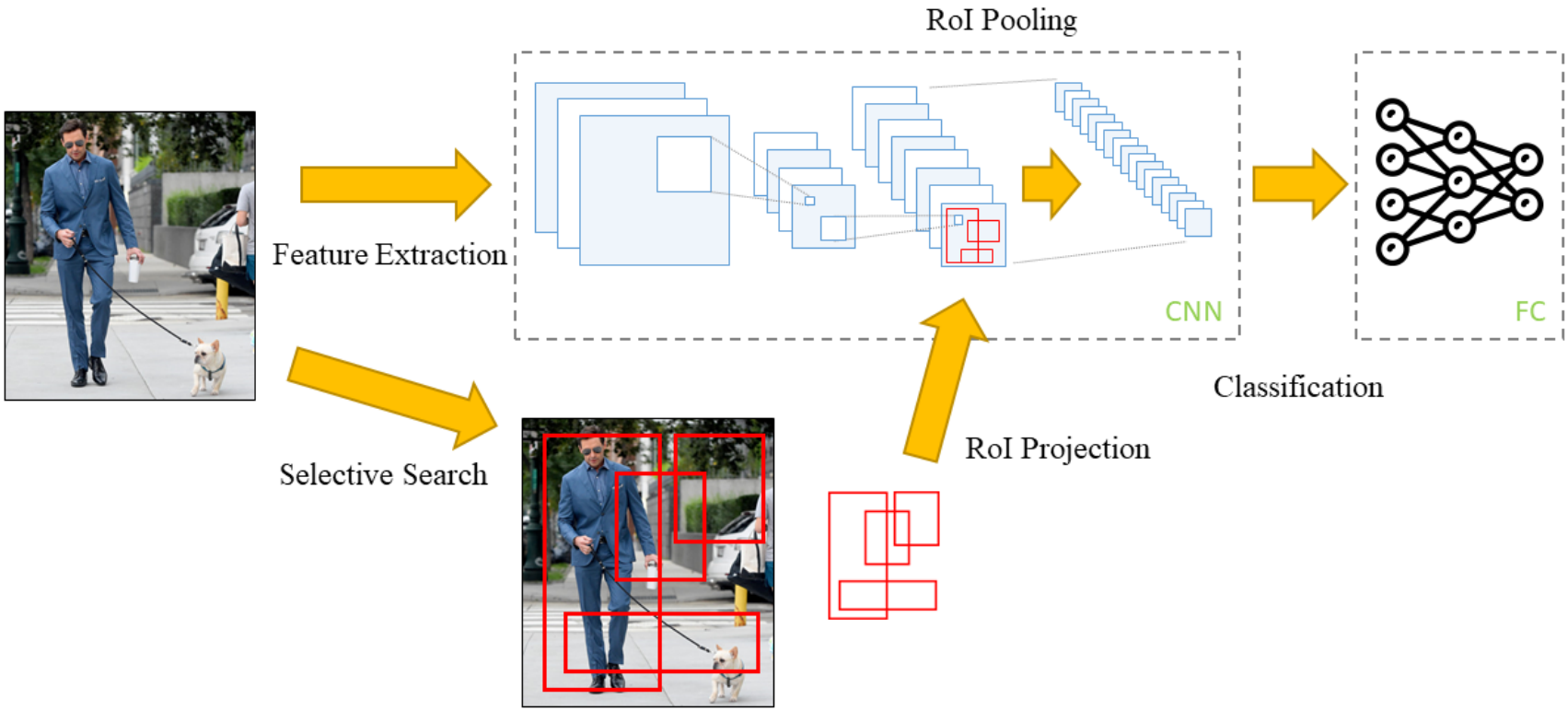


R-CNN

- Disadvantage
  - Owing to about 2000 region proposals needing to be classified in an image, the training process of the network is time-consuming. In summary, R-CNN **can not be used in real-time prediction**.

Fast R-CNN

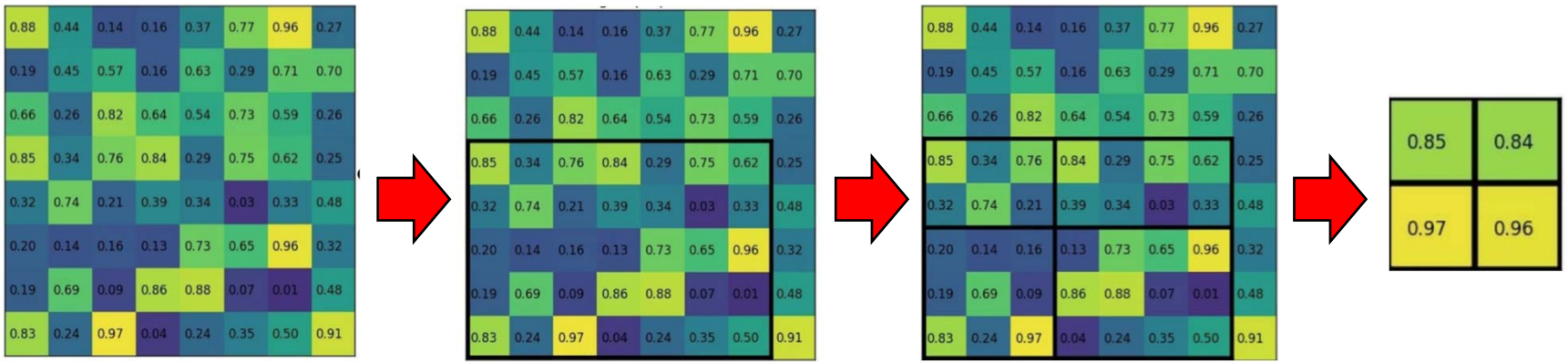
- Compare to R-CNN, only a CNN model is used in an image.



RoI (Region of Interest)

RoI Pooling

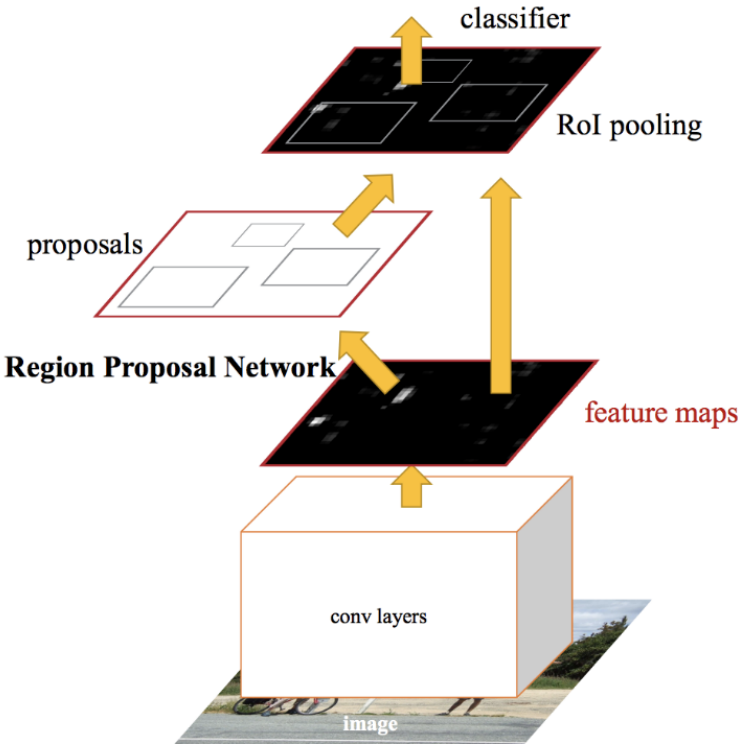
- RoI pooling turns all RoIs into a fixed size because the input size of the fully connected layer is fixed.



Input Region proposal Pooling sections

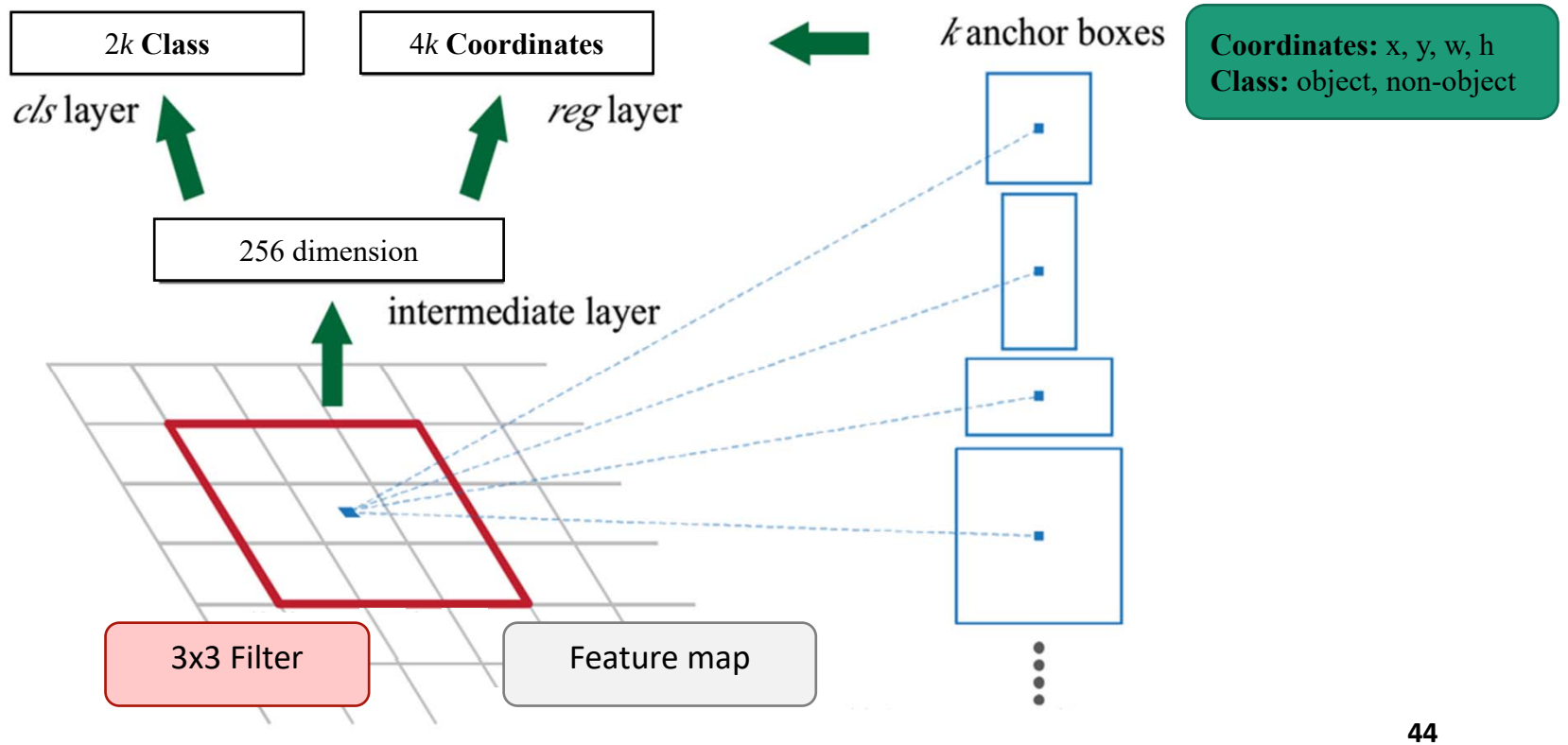
Faster R-CNN

- Process
  - Instead of using selective search in R-CNN and Fast R-CNN, Faster R-CNN use Region Proposal Network (**RPN**) to predict the region proposals.



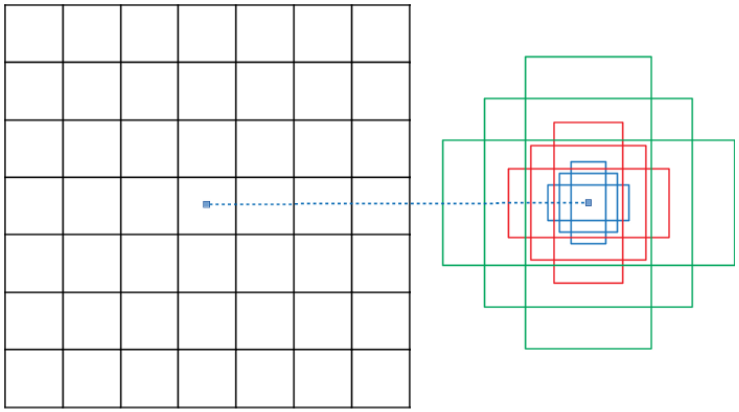
Region Proposal Network (RPN)

- Make CNN do proposals!
  - In Faster-RCNN, RPN replace the algorithm-based region selection method, so GPU acceleration is allowed.

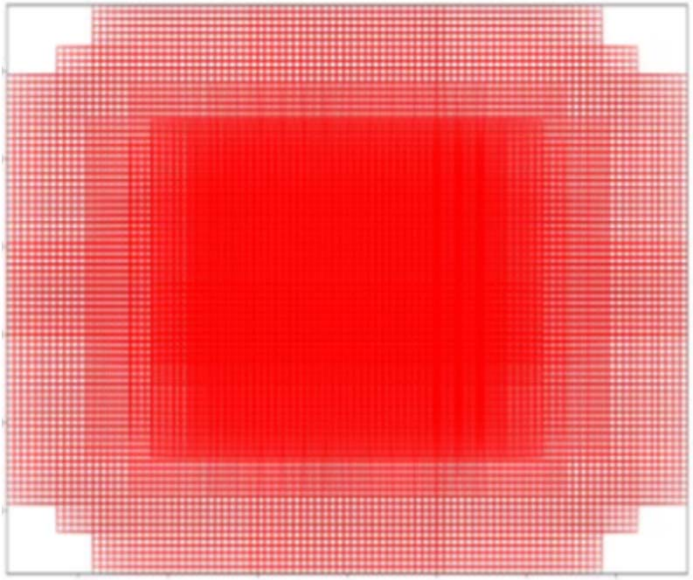


Anchor Boxes

- Faster R-CNN uses anchor boxes with 3 scales and 3 ratios.
- Each pixel on the feature map has 9 anchor boxes.
- The reference boxes are placed at different positions in the image.

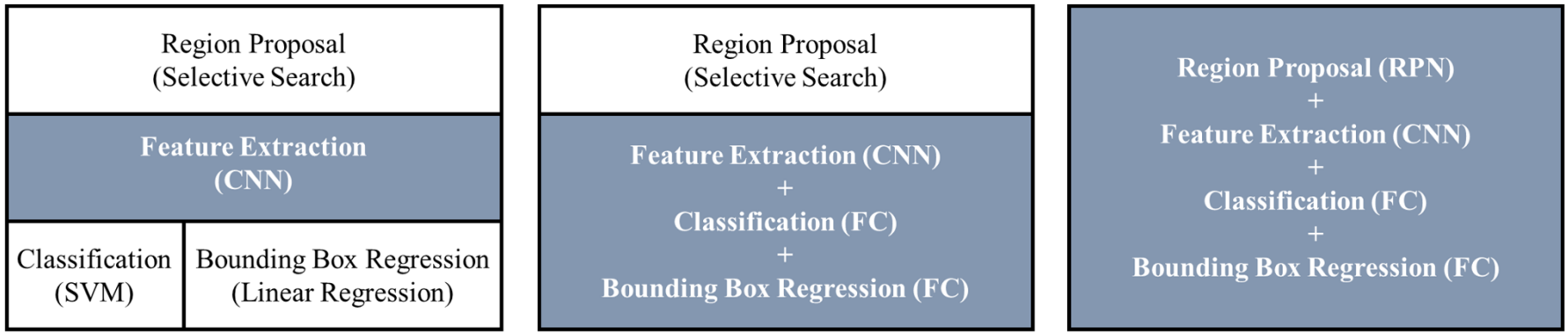


Feature map **Anchor boxes**



Schematic diagram of all anchor boxes mapped to **image**

Comparison



Feature Extractor

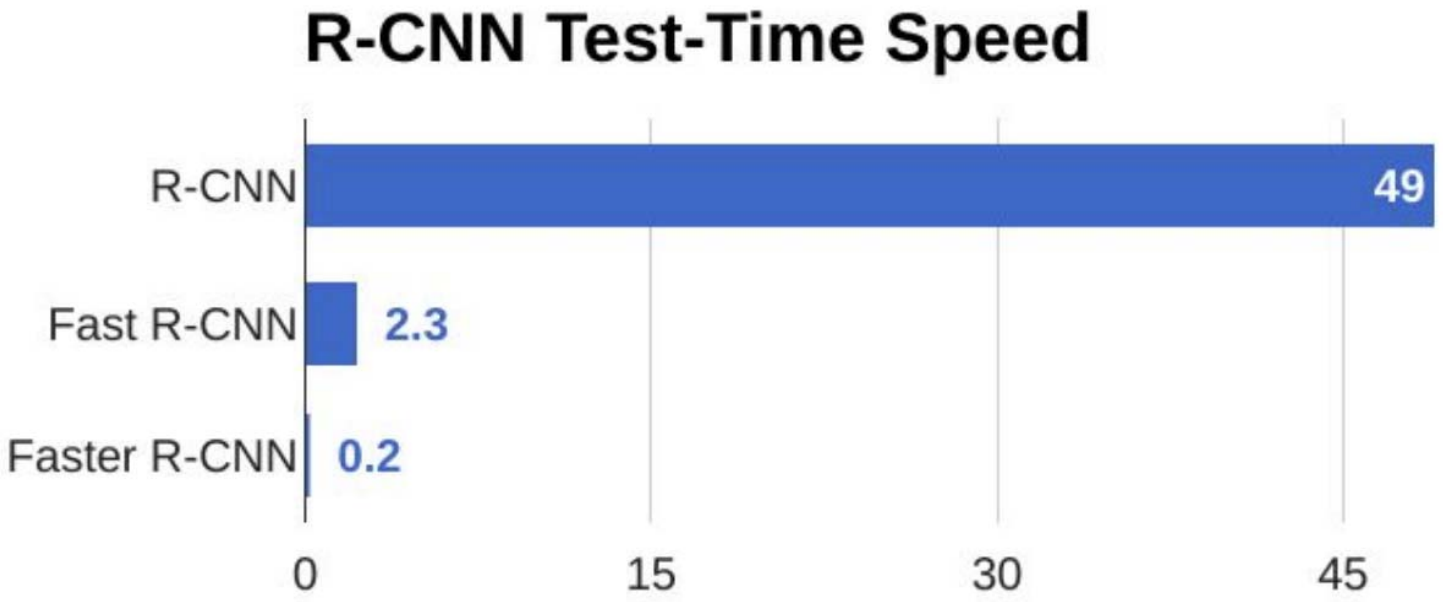
Classification
+
Regression

End-to-End Training

: Deep Learning

Comparison

- Speed comparison for three methods



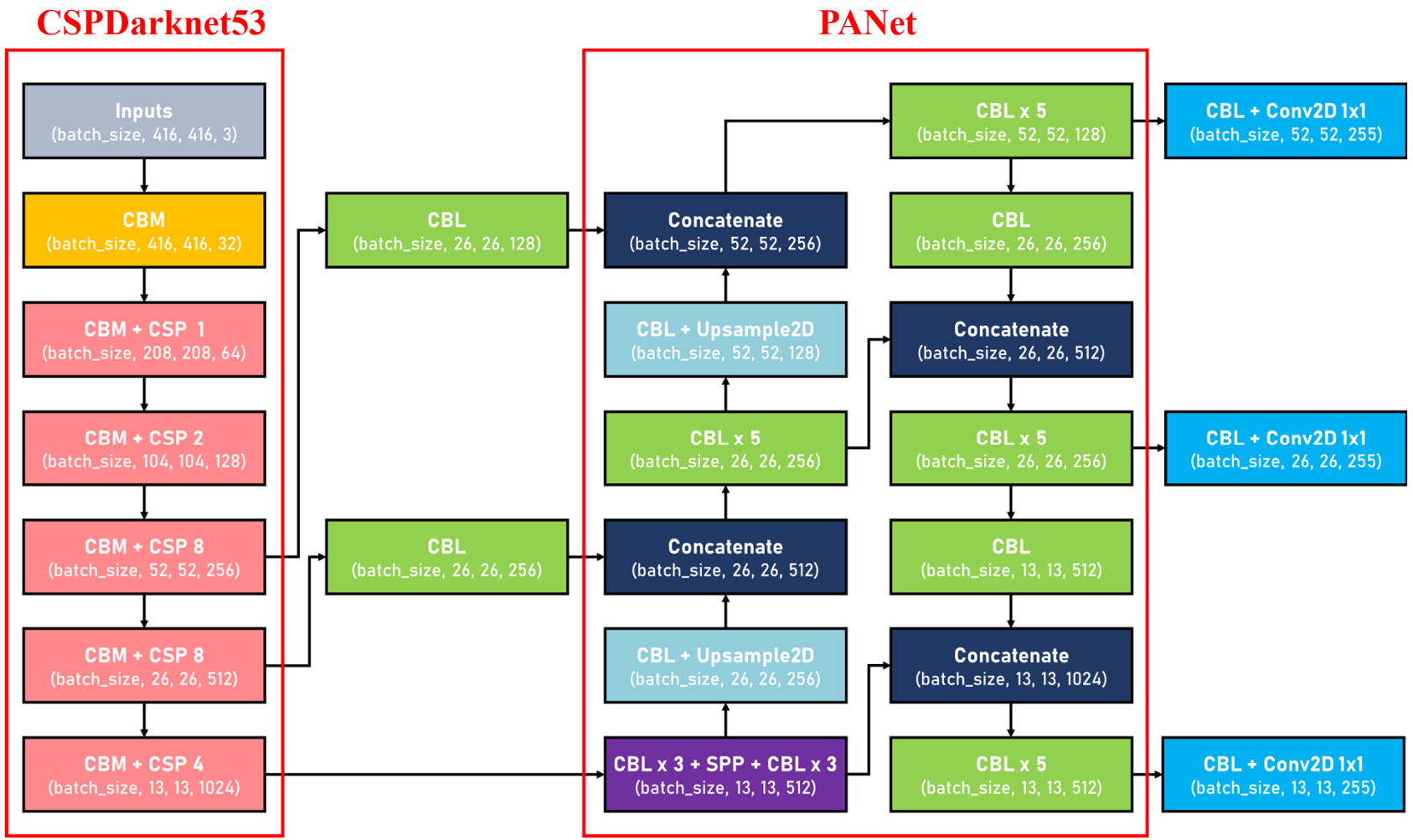
Testing time (seconds) per image

Object Detection with YOLOv4

Outline

- The YOLOv4 Architecture
  - CSPDarknet53
  - Spatial Pyramid Pooling (SPP)
  - Path Aggregation Network (PANet)
- How does the YOLO model work?

Model Architecture

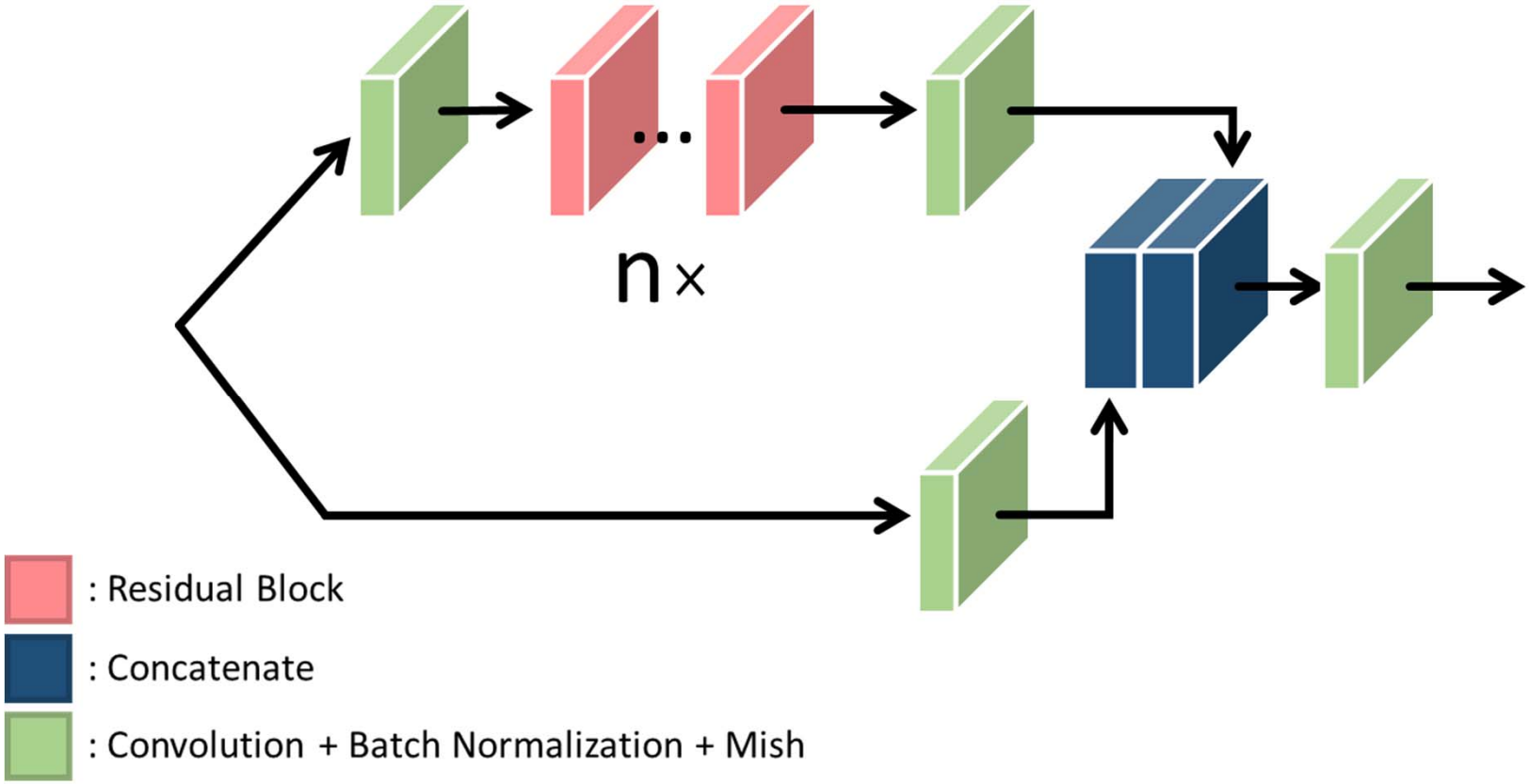


CBM : Convolution + Batch Normalization + Mish

CBL : Convolution + Batch Normalization + Leaky ReLU

CSPDarknet53

- Cross Stage Partial (CSP)
- 能夠在降低20%計算量的情況下保持甚至提高CNN的能力



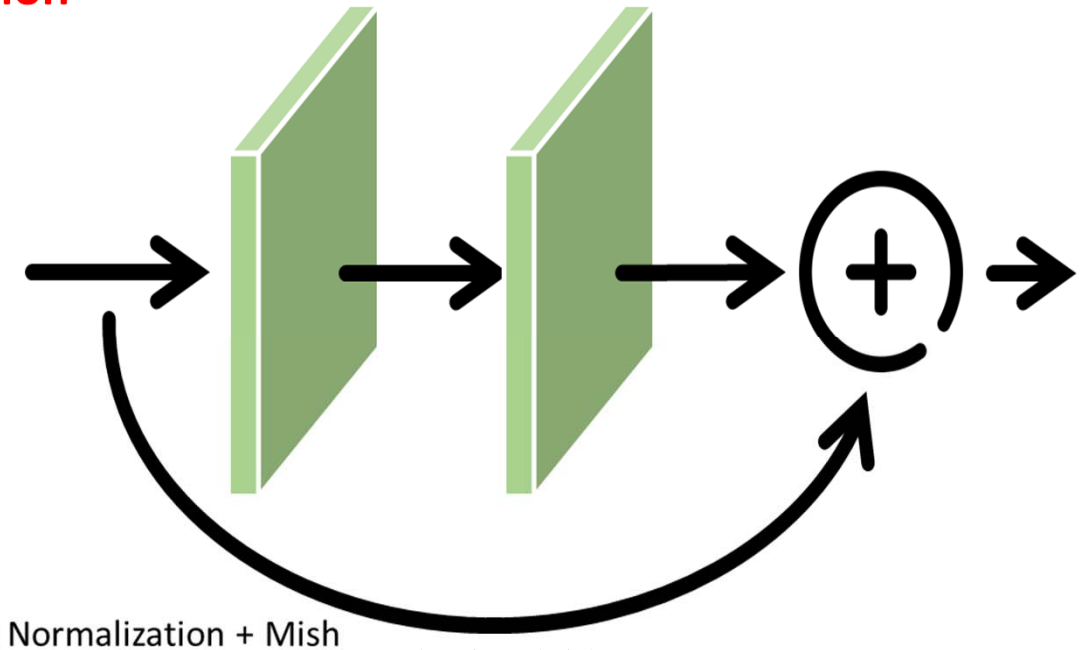
CSPDarknet53

- Cross Stage Partial Network (CSPNet)
  - 從網絡結構設計的角度來解決以往工作在推理過程中需要很大計算量的問題。
  - 作者認爲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。
  - CSPNet通過將梯度的變化從頭到尾地集成到特徵圖中，在減少了計算量的同時可以保證準確率。CSPNet是一種處理的思想，可以和ResNet、ResNeXt和DenseNet結合。

- **Cross Stage Partial (CSP)**
  - **Strengthening the learning ability of a CNN.**
  - **Removing the computational bottlenecks.**
  - **Reducing the memory costs.**

CSPDarknet53

- Residual Block
  - 有一個**捷徑連結**。這個捷徑連結會跳過一層線性轉換和非線性輸出，直接成為上兩層中 activation function 的輸入，因為這個捷徑連結，對來源層的輸出，提供了一個等同於 **identity transform** 或作**直接拷貝**的連結到非相鄰的目標層，減少了一層的轉換，所以又被稱為 **skip connection**。

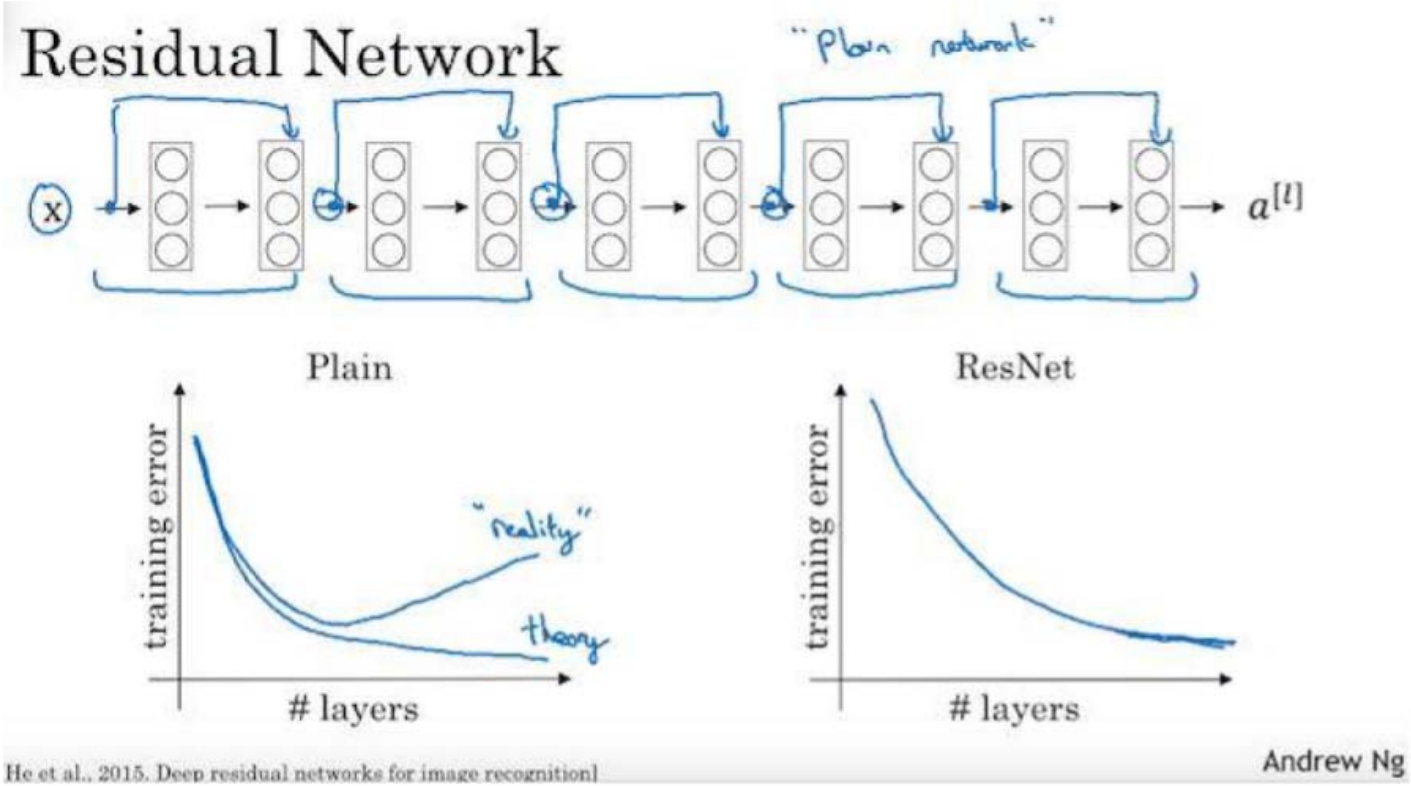


: Convolutional + Batch Normalization + Mish

捷徑連結

Residual Blocks

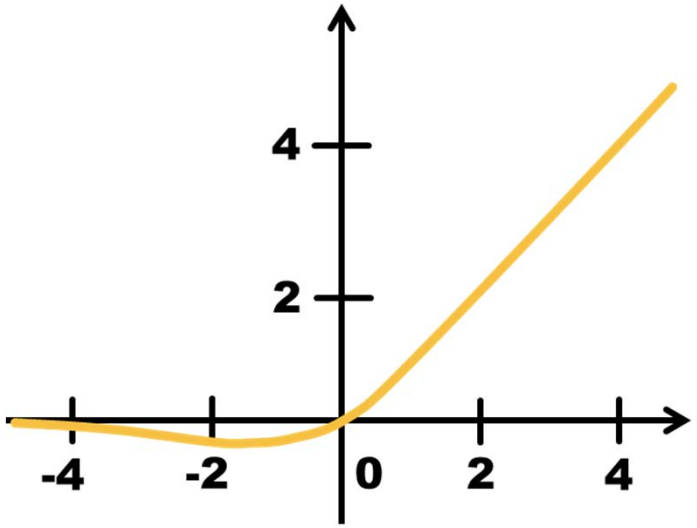
- 可以發現擁有 Residual Blocks 的 Residual Network 在 training error 的表現會逐漸降低 training error（下圖右）而沒有如 plain network 的 training degradation 問題（下圖左，略呈拋物線的曲線，標註為 reality）。



Activation Function

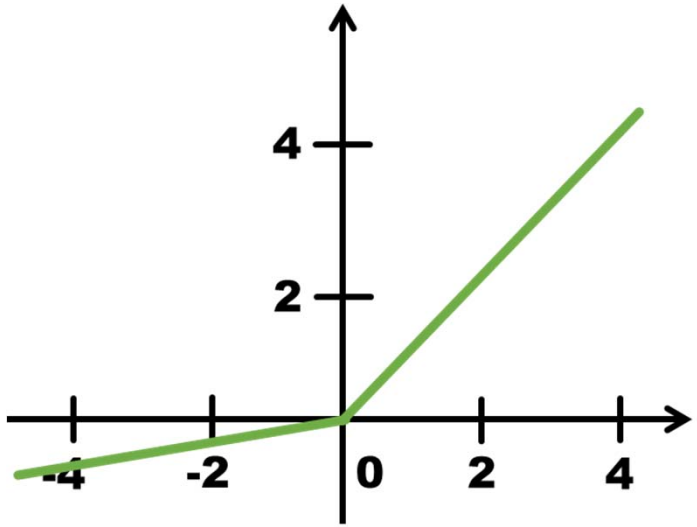
- Owing to the smoothness of **Mish**, it provides better generalization and effective optimization to improve the quality of the results.

$$f(x) = x\,tanh(softplus(x))$$



(a) Mish function

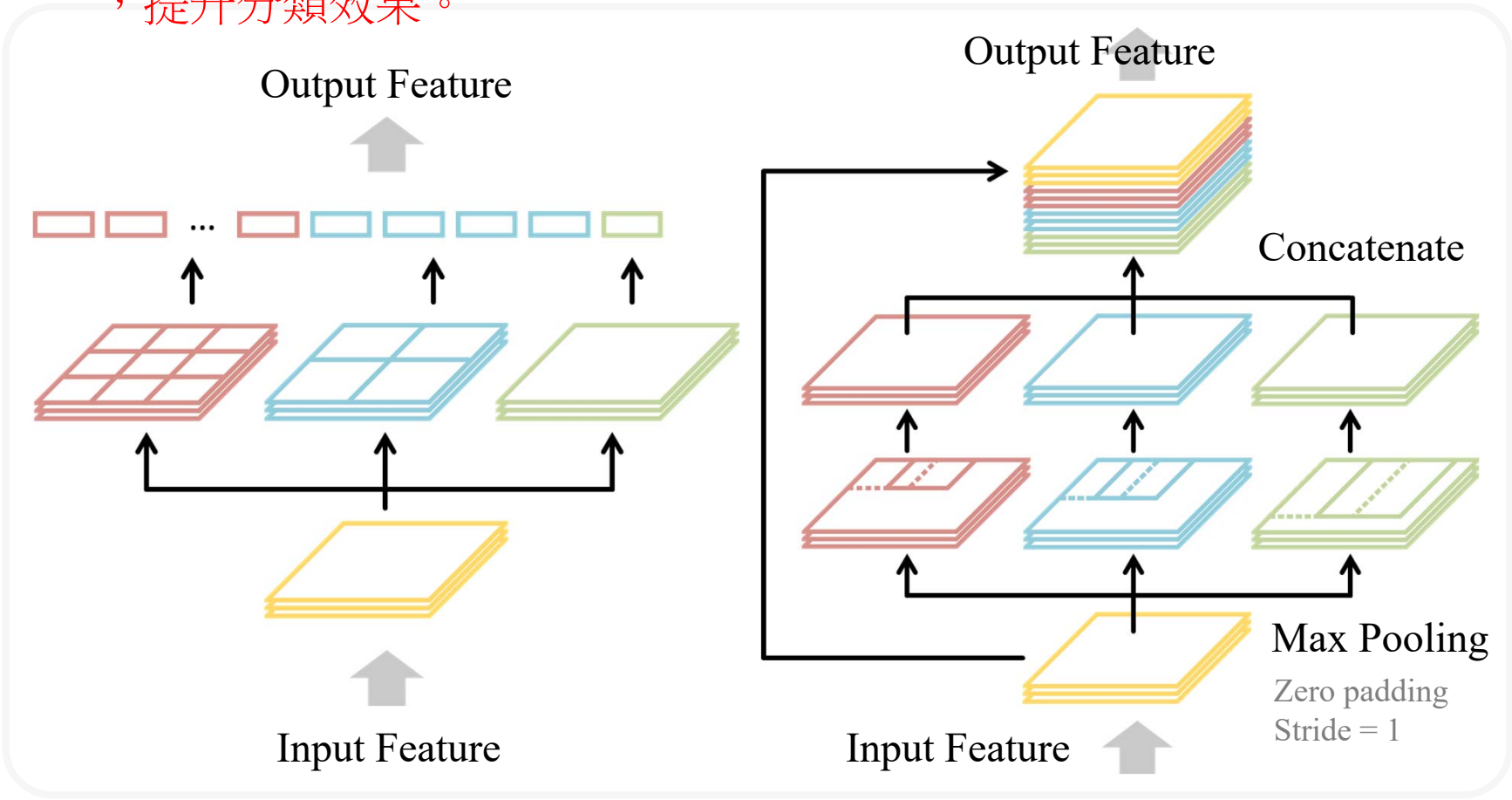
$$f(x) = \max(\frac{x}{a}, x)$$



(b) Leaky ReLU

Spatial Pyramid Pooling (SPP)

是一個處理multi-scale和multi-size靈活的方法，增加CNN的魯棒性，提升分類效果。

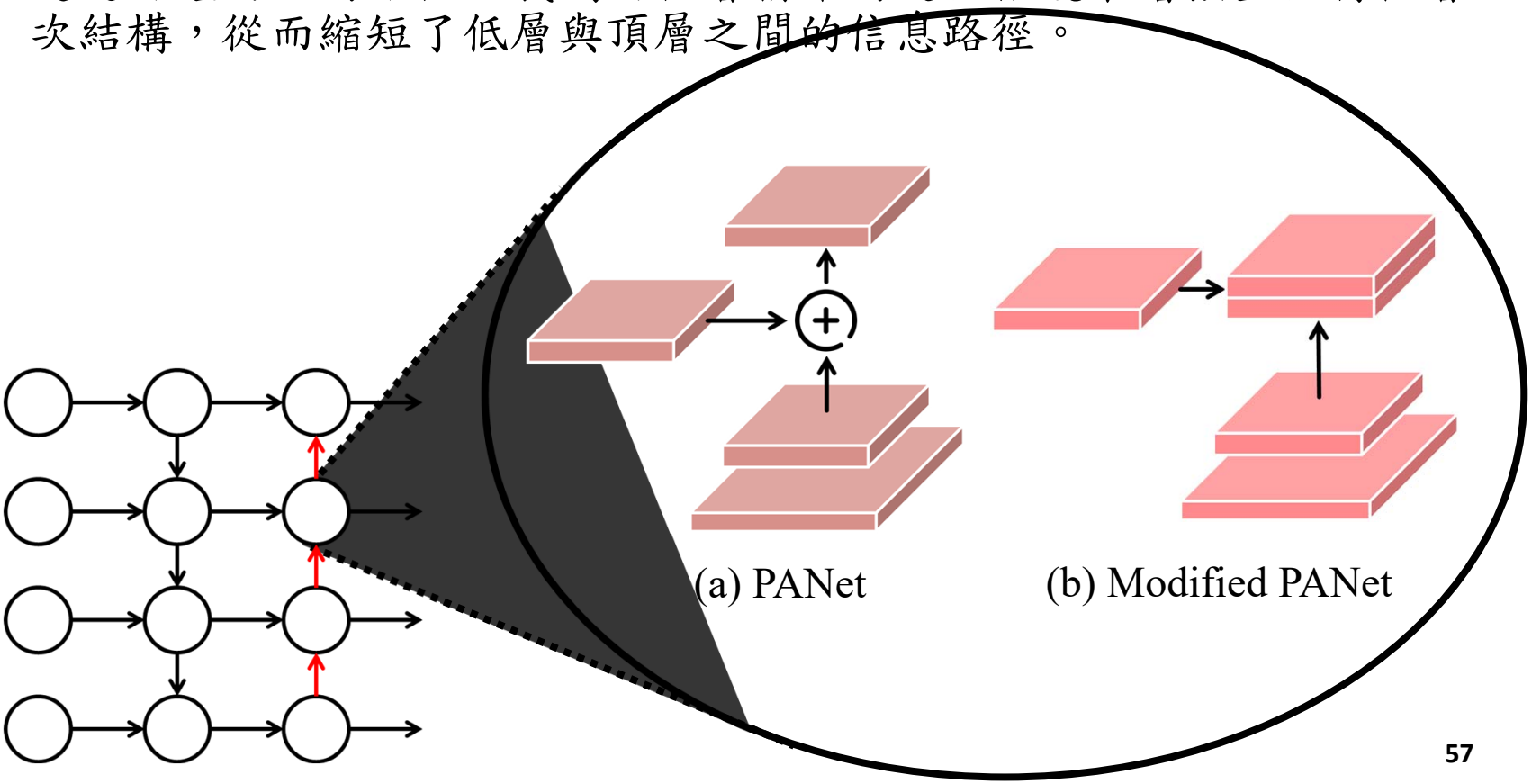


(a) Spatial Pyramid Pooling

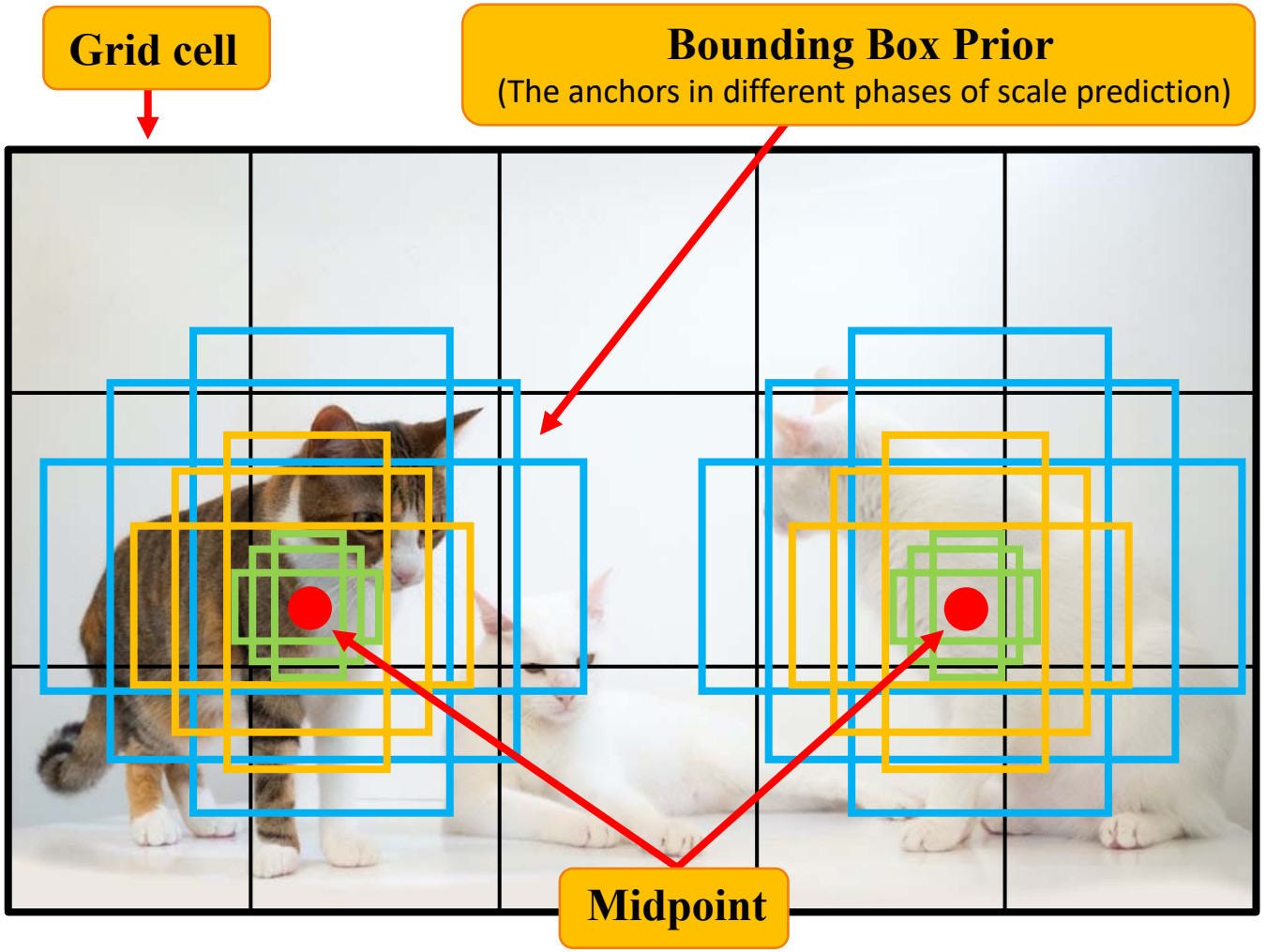
(b) Improved Spatial Pyramid Pooling

Path Aggregation Network (PANet)

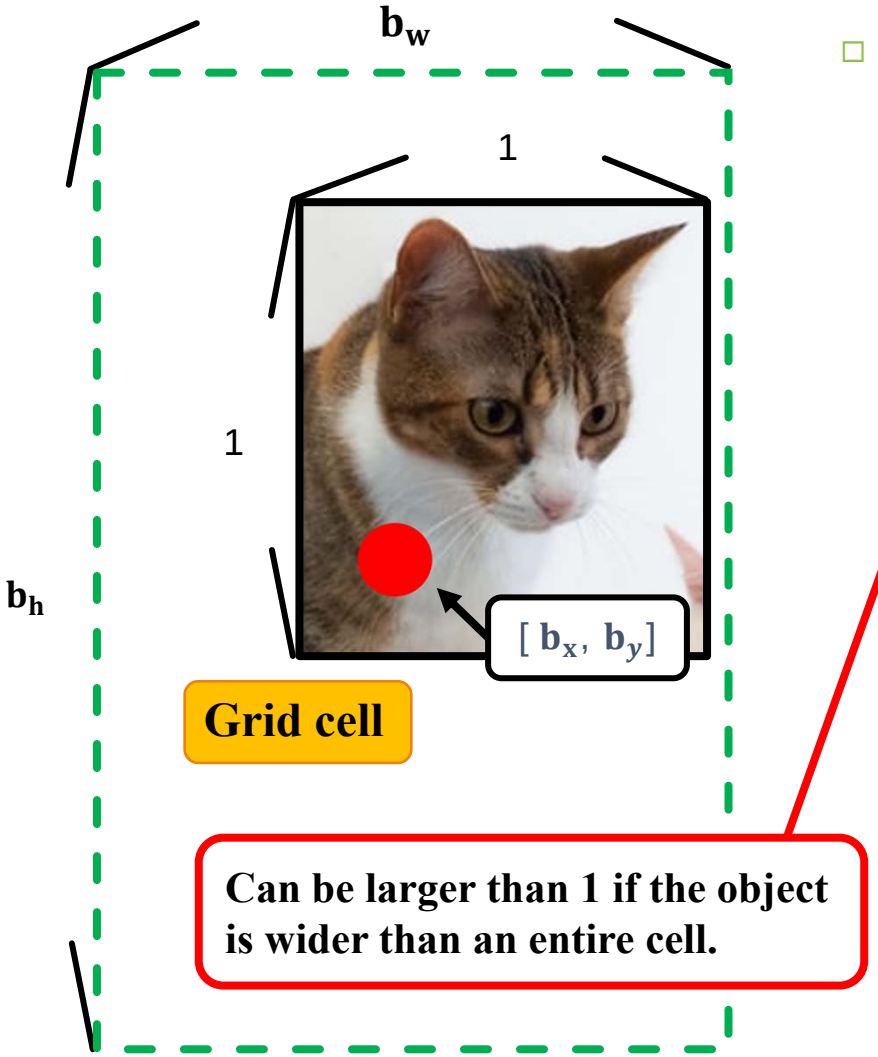
- Bottom-up path augmentation is adopted to preserve the low-level features when the layers become deeper.
- 通過自底向上的路徑，我們用低層精確的定位信號來增強整個特征層次結構，從而縮短了低層與頂層之間的信息路徑。



How does the YOLO model work?



How does the YOLO model work?



- Each bounding box has four values :

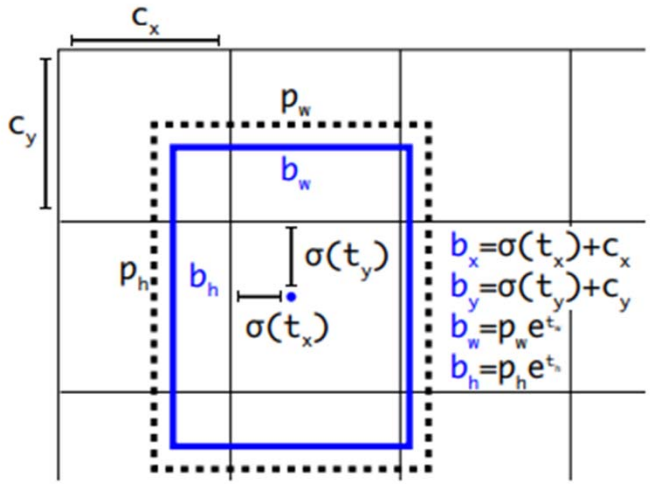
  〔 b_x , b_y , b_w , b_h 〕

  b_x : Position x

  b_y : Position y

  b_w : width of the bounding box

  b_h : height of the bounding box



Scale Prediction

- The size of the output feature map in each phase of scale prediction is 13×13, 26×26, and 52×52.
- Scale prediction helps the YOLO model to make possible detection and detect the smaller object.
- Each grid cell will make a detection for three bounding boxes with the output :

  → (Probability, t_x, t_y, t_w, t_h, class)

Total predictions = (13×13 + 26×26 + 52×52) × 3 × (5 + class) = 266,175 predictions

Loss Function

$$\mathcal{L}_{bbox} = \lambda_{crood} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{i,j}^{obj} (1 - CIoU(pred, truth))$$

CIoU-loss

$$\mathcal{L}_{obj} = \lambda_{obj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{i,j}^{obj} [-C_{ij} \log \hat{C}_{ij} - (1 - C_{ij}) \log(1 - \hat{C}_{ij})]$$

Cross-entropy Loss

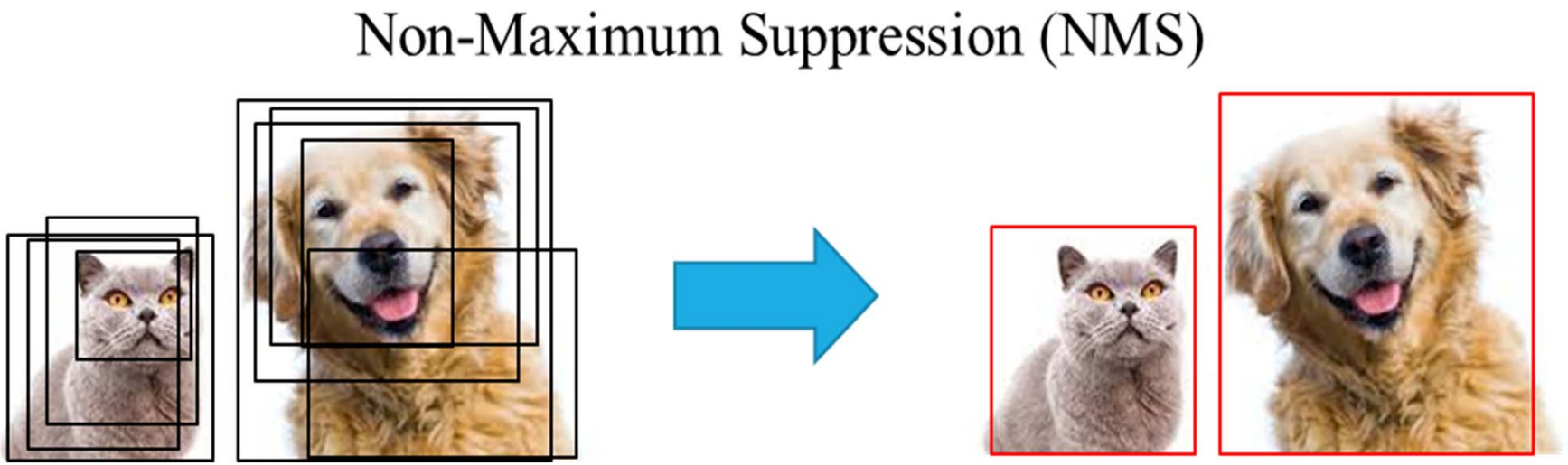
$$\mathcal{L}_{cls} = \lambda_{class} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{i,j}^{obj} \sum_{c \epsilon classes} [-p_i(c) \log \hat{p}_i(c) - (1 - p_i(c)) \log(1 - \hat{p}_i(c))]$$

$$\mathcal{L}_{total} = \mathcal{L}_{bbox} + \mathcal{L}_{cls} + \mathcal{L}_{obj}$$

S = Scale Prediction Size → 13×13, 26×26, 52×52

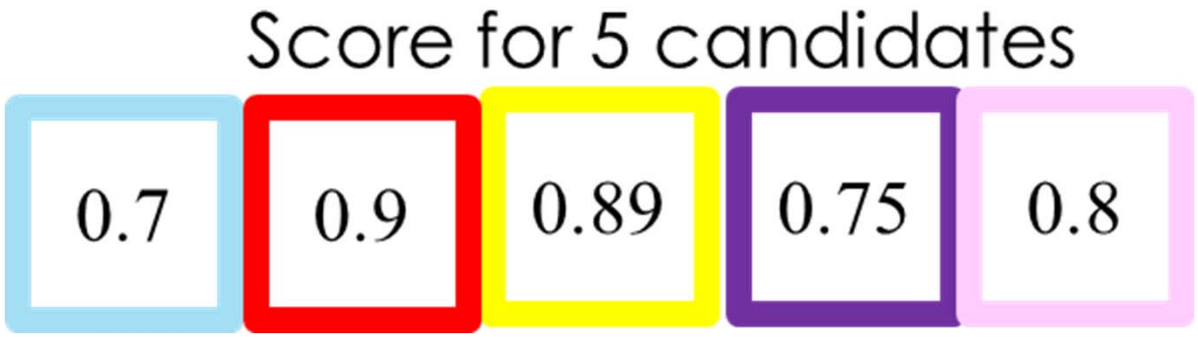
Non-Maximum Suppression (NMS)

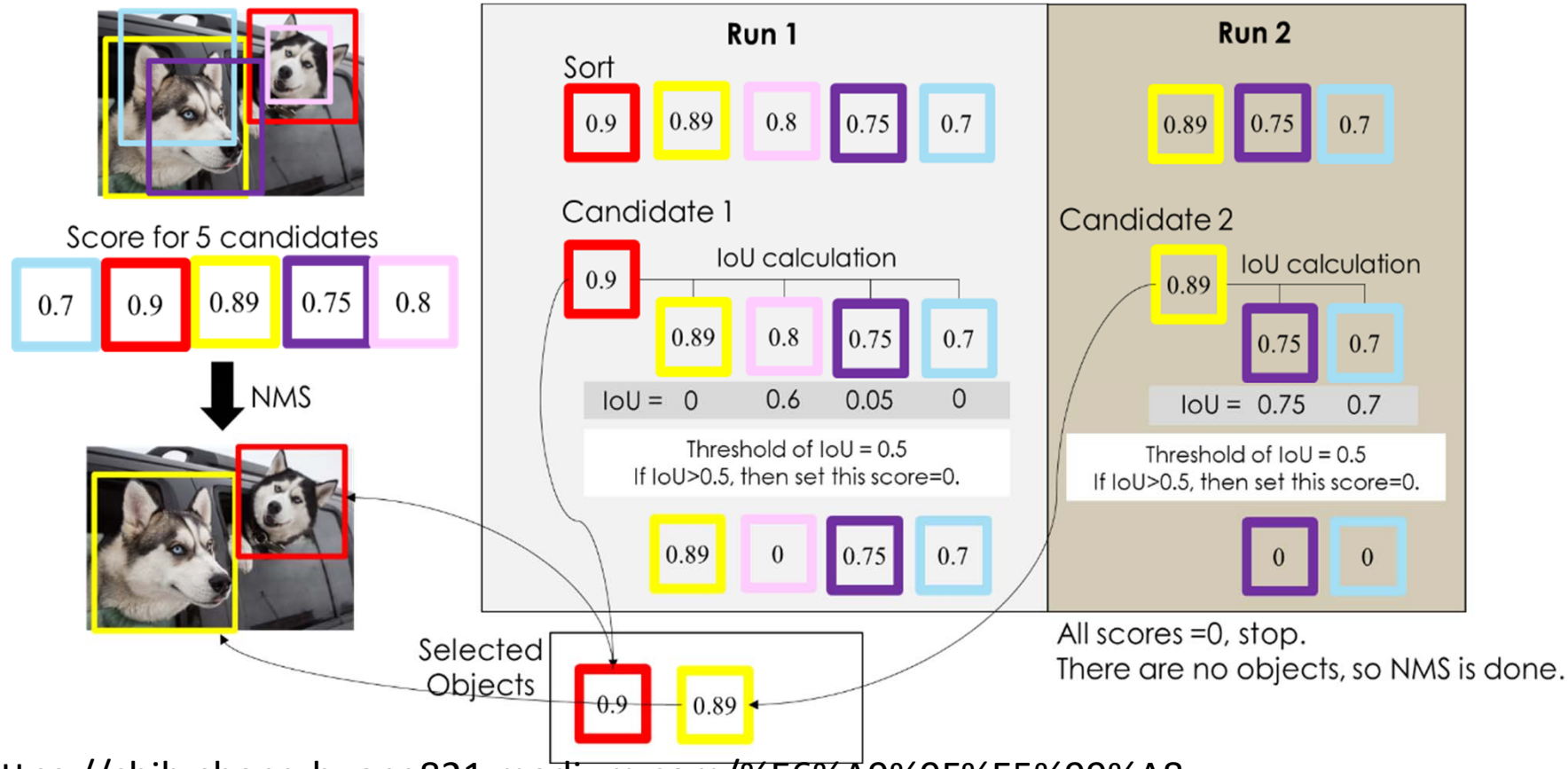
- 有可能一個物件被很多候選框給選到，下左圖假設算法抓到這麼多框都是物件，這時候要怎麼處理，用Non-Maximum Suppression的方法去消除多餘物件框找到最佳的框，



Algorithm

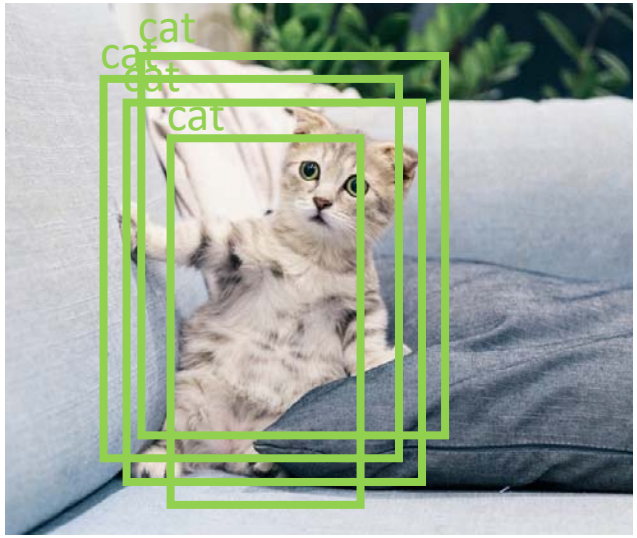
- 先看哪個BBox的信心程度最高，那個BBox會進去「確定是物件集合」(途中的selected objects)內
- 其他BBox和剛選出來的BBox算IoU，然後算出來的IoU大於設定好的閾值的BBox，那些BBox的信心度會被設定為0(也就是這個BBox重複計算要刪掉)。

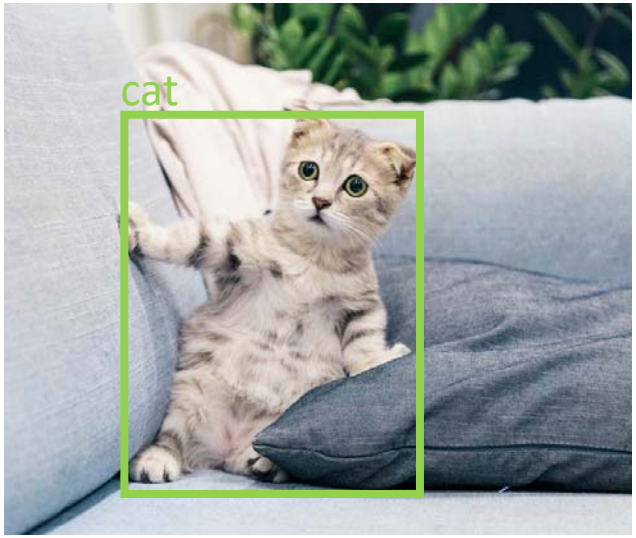




https://chih-sheng-huang821.medium.com/%E6%A9%9F%E5%99%A8-%E6%B7%B1%E5%BA%A6%E5%AD%B8%E7%BF%92-%E7%89%A9%E4%BB%B6%E5%81%B5%E6%B8%AC-non-maximum-suppression-nms-aa70c45a

Non-Maximum Suppression (NMS)





NMS

$$S_i = \begin{cases} s_i, & IoU(M, b_i) < N_t \\ 0, & IoU(M, b_i) \geq N_t \end{cases}$$

DIoU-NMS

$$S_i = \begin{cases} s_i, & IoU - R_{DIoU}(M, B_i) < \varepsilon \\ 0, & IoU - R_{DIoU}(M, B_i) \geq \varepsilon \end{cases}$$

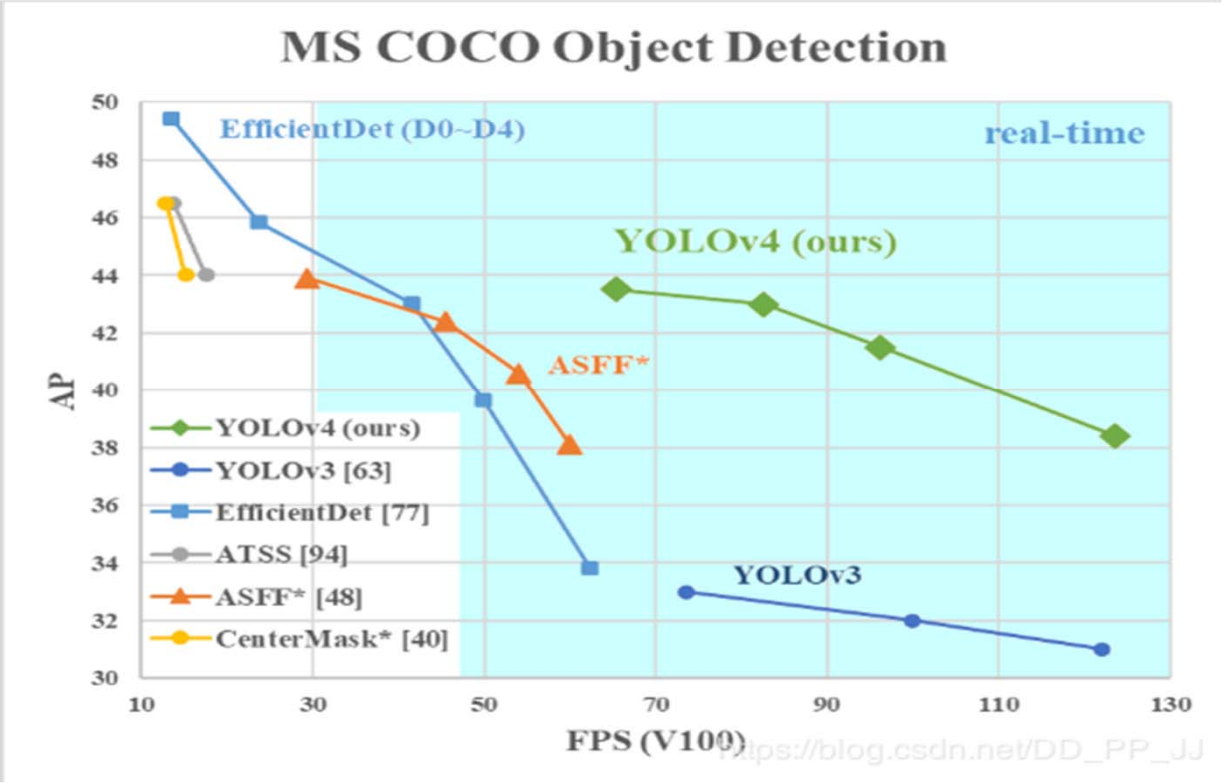
Yolov4 下載與安裝

YOLOv4-Alexey Bochkovskiy 版

- 可以應用在工廠瑕疵檢測、醫療影像分析、生物影像分析、工安影像分析、口罩影像分析等。
- YOLO Darknet 的維護者俄羅斯人 **Alexey Bochkovskiy** 發現中研院資科所博後**王建堯**及所長**廖弘源**研發的 CSPDarkNet-53 detector 又快又好，於是邀請中研院資科所以此為 backbone 發展 YOLOv4 ，速度及正確率比上一代 YOLOv3 都要好。
- 最終得到YOLO**官方Joseph Redmon**的認同，成了第四代接棒者。
- YOLOv4 英文論文： https://arxiv.org/abs/2004.10934
- YOLOv4 英文論文中文翻譯： https://bangqu.com/2166Wr.html
- YOLOv4 開放原始碼： https://github.com/AlexeyAB/darknet
- CSPNet 英文論文： https://arxiv.org/pdf/1911.11929.pdf
- CSPNet 開放原始碼： https://github.com/WongKinYiu/CrossStagePartialNetworks
- 影片介紹 https://www.youtube.com/watch?time_continue=1&v=AbsSrqFkOb0&feature=emb_logo

Performance

- YOLOv4屬於One-Stage (SSD) 的Object Detection演算法。
  - 一般來說，AP(Average Precision)會比Two-Stage (如：R-CNN、Fast R-CNN、Faster R-CNN....系列) 低，但相對的FPS的表現會比較好一些。 在COCO dataset上，43.5% AP，速度 65 FPS！ (Tesla V100 GPU)
  - FPS優於Two-Stage外，甚至都超越了Two-Stage的演算法。



Requirements

- **CMake >= 3.18**:
- **CUDA 10.2**:
- **OpenCV >= 2.4 cuDNN >= 8.0.2**
- **GPU with CC >= 3.0**

Yolo v4 in other frameworks

- **TensorFlow: YOLOv4 on TensorFlow 2.0 / TFlite / Andriod**:
- **OpenCV-dnn** the fastest implementation of **YOLOv4 for CPU Tencent/ncnn**:
  - the fastest inference of YOLOv4 on mobile phone CPU: https://github.com/Tencent/ncnn
- **PyTorch > ONNX > CoreML > iOS** how to convert cfg/weights-files to pt-file: ultralytics/yolov3 and iOS App
- **TensorRT YOLOv4 on TensorRT+tkDNN**:
  - https://github.com/ceccocats/tkDNN For YOLOv3 (-70% faster inference):
  - Yolo is natively supported in DeepStream 4.0
  - wang-xinyu/tensorrtx implemented yolov3-spp, yolov4, etc.

More links

- Pytorch implementation
  - https://github.com/GZQ0723/YoloV4
  - https://github.com/Tianxiaomo/pytorch-YOLOv4

- Keras:
  - https://github.com/Ma-Dan/keras-yolo4

- YOLOv4-QtGUI Windows 10環境下，
  - YOLOv4-QtGUI是用QT和OpenCV開發視覺化目標檢測介面,可簡單選擇本地圖片、或攝像頭輸入來展示檢測結果。
  - https://github.com/scutlrr/Yolov4-QtGUI

YOLOv4 下載安裝

- 原始碼下載
  - https://github.com/AlexeyAB/darknet
  - 含安裝文件

- Windows版本安裝文件 **2020/05/21**
  - https://ithelp.ithome.com.tw/articles/10231508
  - 採用vcpkg 建置

- **在windows安裝YOLO darknet – GPU 2020/06/05**
  - https://ithelp.ithome.com.tw/articles/10231950

YOLOv4 on Jetson Nano install

- https://medium.com/@thundo/yolov4-on-jetson-nano-672c1d38aed2
- 安裝指令
  - **$ sudo apt update**
  - **$ sudo apt install -y git wget build-essential gcc g++ make binutils libcanberra-gtk-module**
  - $ mkdir ~/Projects #視需要
  - $ cd ~/Projects #視需要
  - $ echo "export PATH=/usr/local/cuda/bin${PATH:+:${PATH}}" >> ~/.bashrc
  - $ echo "export LD_LIBRARY_PATH=/usr/local/cuda/lib64${LD_LIBRARY_PATH:+:${LD_LIBRARY_PATH}}" >> ~/.bashrc
  - $ **source ~/.bashrc**
  - **$ git clone https://github.com/AlexeyAB/darknet.git**
  - $ **cd darknet**
  - $ **vim Makefile**

Makefile setting 說明

- Just do make in the darknet directory.
  - GPU=1 to build with CUDA to accelerate by using GPU (CUDA should be in **/usr/local/cuda**)
  - CUDNN=1 to build with cuDNN v5-v7 to accelerate training by using GPU (cuDNN should be in **/usr/local/cudnn**)
  - CUDNN_HALF=1 to build for Tensor Cores (on Titan V / Tesla V100 / DGX-2 and later) speedup Detection 3x, Training 2x
  - OPENCV=1 to build with OpenCV 4.x/3.x/2.4.x - allows to detect on video files and video streams from network cameras or web-cams
  - DEBUG=1 to build debug version of Yolo
  - OPENMP=1 to build with OpenMP support to accelerate Yolo by using multi-core CPU
  - **LIBSO=1 to libdarknet.so**

安裝

- 修改Makefile **($vim Makefile)**設定 for jetson nano
  - **GPU=1**
  - **CUDNN=1**
  - **CUDNN_HALF=1**
  - **OPENCV =1**
  - **LIBSO=1 …**
  - **ARCH= -gencode arch=compute_53,code=[sm_53,compute_53]**
  - **此值必須參考顯卡計算力**
- Save, exit and start compiling with
  - **$ make -j4**
- **產生**
  - **darknet 與 libdarknet.so**
- **可能問題 nvcc not found**
  - **確認nvcc位置 (可能在 /usr/local/cuda/bin)**
  - **$ echo "PATH=/usr/local/cuda/bin:$PATH" >> ~/.bashrc**
  - **$ source ~/.bashrc**

https://jkjung-avt.github.io/yolov4/

Setting for yolov4

- Download the pretrained model weights
  - **$ wget https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.weights**
  - **課程ftp站下載 ftp://120.107.172.222/yolov4/yolov4.weights 247MB**

- Edit width/height parameters in the cfg file
  - $ **vim ~/darknet/cfg/yolov4.cfg**
- and set them both to 224
  - **width=224**
  - **height=224**
- 安裝目錄在 **~/darknet** 之下 執行前要切換目錄

執行指令範例 ./darknet

- **Yolo v4 COCO - image:**
  - **$ ./**darknet detector test ./cfg/coco.data ./cfg/yolov4.cfg ./yolov4.weights -thresh 0.25
  - 會要求輸入**image路徑與檔案**，請輸入 **./data/dog.jpg or eagle.jpg**
  - 結果會存在 **./darknet/prediction.jpg**
- **Output coordinates of objects:**
  - **$ ./**darknet detector test **./**cfg/coco.data ./cfg/yolov4.cfg ./yolov4.weights -ext_output ./data/dog.jpg
  - 輸出座標
- **Yolo v4 COCO - video:**
  - **$** ./darknet detector **demo** ./cfg/coco.data ./cfg/yolov4.cfg yolov4.weights -ext_output ./data/test.mp4
- 請注意用jtop 觀察GPU是否有運作
- 若沒有opencv可能沒有安裝支援cuda cudnn (要再重裝) 或是沒有連結好

執行範例

- **Yolo v4 COCO - WebCam 0:**
  - **$ ./**darknet detector **demo** ./cfg/coco.data ./cfg/yolov4.cfg yolov4.weights -c 0

- **Yolo v4 COCO for net-videocam - Smart WebCam:**
  - **$ ./**darknet detector **demo** ./cfg/coco.data ./cfg/yolov4.cfg yolov4.weights http://192.168.0.80:8080/video?dummy=param.mjpg

- **Yolo v4 - save result videofile res.avi:**
  - **$ ./**darknet detector **demo** ./cfg/coco.data ./cfg/yolov4.cfg yolov4.weights ./data/test.mp4 -out_filename res.avi

YOLOv4 教學影片

- 教學網頁
  - https://tw.openrobot.org/article/index?sn=11716
- Youtube 影片
  - https://www.youtube.com/watch?time_continue=4&v=m-PFX_e5HFk&feature=emb_logo
- YOLOv4 多物件追蹤
  - https://tw.openrobot.org/article/index?sn=11782
  - 影片 https://www.youtube.com/watch?time_continue=1&v=cN_mB-P1PBk&feature=emb_logo

Demo

- YOLOv4 - The most accurate real-time neural network for object detection
  - https://www.youtube.com/watch?v=1_SiUOYUoOI
- YOLO v3 vs YOLO v4
  - https://www.youtube.com/watch?v=q9f_e1nc4wc
- Yolov3 v4 比較
  - https://www.youtube.com/watch?time_continue=14&v=m-PFX_e5HFk&feature=emb_logo
- YOLOV3-Tiny, YOLOV3, YOLOV4, YOLOV5-s
  - https://www.youtube.com/watch?v=Ri7XTxK7EYw&feature=emb_logo

Related issues

- **tkDNN-TensorRT** accelerates YOLOv4 **~2x** times for batch=1 and **3x-4x** times for batch=4.
- **OpenCV-dnn** is ~10% slower than tkDNN-TensorRT.
  - tkDNN: https://github.com/ceccocats/tkDNN
  - OpenCV: https://gist.github.com/YashasSamaga/48bdb167303e10f4d07b754888ddbdcf

Network Size	Darknet, FPS (avg)	tkDNN TensorRT FP32, FPS	tkDNN TensorRT FP16, FPS	OpenCV FP16, FPS	tkDNN TensorRT FP16 batch=4, FPS	OpenCV FP16 batch=4, FPS	tkDNN Speedup
320	100	116	**202**	171	**423**	384	**4.2x**
416	82	103	**162**	146	**284**	260	**3.5x**
512	69	91	**134**	125	**206**	190	**2.9x**
608	53	62	**103**	100	**150**	133	**2.8x**

More Pre-trained models

There are weights-file for different cfg-files (trained for MS COCO dataset):

FPS on RTX 2070 (R) and Tesla V100 (V):

- yolov4.cfg - 245 MB: yolov4.weights (Google-drive mirror yolov4.weights) paper Yolo v4 just change `width=` and `height=` parameters in `yolov4.cfg` file and use the same `yolov4.weights` file for all cases:
  - `width=608 height=608` in cfg: **65.7% mAP@0.5 (43.5% AP@0.5:0.95) - 34(R) FPS / 62(V) FPS** - 128.5 BFlops
  - `width=512 height=512` in cfg: **64.9% mAP@0.5 (43.0% AP@0.5:0.95) - 45(R) FPS / 83(V) FPS** - 91.1 BFlops
  - `width=416 height=416` in cfg: **62.8% mAP@0.5 (41.2% AP@0.5:0.95) - 55(R) FPS / 96(V) FPS** - 60.1 BFlops
  - `width=320 height=320` in cfg: **60% mAP@0.5 (38% AP@0.5:0.95) - 63(R) FPS / 123(V) FPS** - 35.5 BFlops
- yolov3-tiny-prn.cfg - **33.1% mAP@0.5 - 370(R) FPS** - 3.5 BFlops - 18.8 MB: yolov3-tiny-prn.weights
- enet-coco.cfg (EfficientNetB0-Yolov3) - **45.5% mAP@0.5 - 55(R) FPS** - 3.7 BFlops - 18.3 MB: enetb0-coco_final.weights
- yolov3-openimages.cfg - 247 MB - 18(R) FPS - OpenImages dataset: yolov3-openimages.weights

課程**ftp**站下載 **ftp://120.107.172.222/yolov4/**

Pre-trained models (updated)

FPS on RTX 2070 (R) and Tesla V100 (V):

- yolov4-p6.cfg - 1280x1280 - **72.1% mAP@0.5 (54.0% AP@0.5:0.95) - 32(V) FPS** - xxx BFlops (xxx FMA) - 487 MB: yolov4-p6.weights
    - pre-trained weights for training: https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v4_pre/yolov4-p6.conv.289
- yolov4-p5.cfg - 896x896 - **70.0% mAP@0.5 (51.6% AP@0.5:0.95) - 43(V) FPS** - xxx BFlops (xxx FMA) - 271 MB: yolov4-p5.weights
    - pre-trained weights for training: https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v4_pre/yolov4-p5.conv.232
- yolov4-csp-x-swish.cfg - 640x640 - **69.9% mAP@0.5 (51.5% AP@0.5:0.95) - 23(R) FPS / 50(V) FPS** - 221 BFlops (110 FMA) - 381 MB: yolov4-csp-x-swish.weights
    - pre-trained weights for training: https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v4_pre/yolov4-csp-x-swish.conv.192

Pre-trained models

- yolov4-csp-swish.cfg - 640x640 - **68.7% mAP@0.5 (50.0% AP@0.5:0.95) - 70(V) FPS** - 120 (60 FMA) - 202 MB: yolov4-csp-swish.weights
    - pre-trained weights for training: https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v4_pre/yolov4-csp-swish.conv.164
- yolov4x-mish.cfg - 640x640 - **68.5% mAP@0.5 (50.1% AP@0.5:0.95) - 23(R) FPS / 50(V) FPS** - 221 BFlops (110 FMA) - 381 MB: yolov4x-mish.weights
    - pre-trained weights for training: https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v4_pre/yolov4x-mish.conv.166
- yolov4-csp.cfg - 202 MB: yolov4-csp.weights paper Scaled Yolo v4

  just change `width=` and `height=` parameters in `yolov4-csp.cfg` file and use the same `yolov4-csp.weights` file for all cases:

    - `width=640 height=640` in cfg: **67.4% mAP@0.5 (48.7% AP@0.5:0.95) - 70(V) FPS** - 120 (60 FMA) BFlops
    - `width=512 height=512` in cfg: **64.8% mAP@0.5 (46.2% AP@0.5:0.95) - 93(V) FPS** - 77 (39 FMA) BFlops
    - pre-trained weights for training: https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v4_pre/yolov4-csp.conv.142

YOLOv4 Python Example

- Please reference/trace
  - darknet.py
  - darknet_images.py
  - darknet_video.py

訓練前準備

Why Image Labeling?

- 目的：
  - 讓電腦能夠透過學習來認知影像中的物件(電腦視覺)
- 用途：
  - 自駕車、機器人、相片搜尋、影片檢索、商品搜尋…

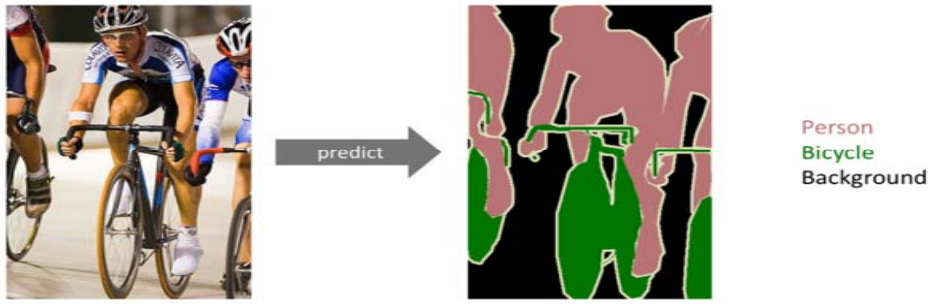
- Training (or Testing) examples for Supervised Learning

影像辨識的種類及 labeling

- **影像分類(Classification)**：
  - 最基本的辨識，輸入整張圖片後由電腦判斷其的類別，相片不需要label但需要作分類。Dog, cat, truck (0/1, or classnames)
- **分類＋定位(Classification + Localization)**：
  - 除了看出圖片中的物體，電腦需要能知道該物體在圖片中的位置
  - 需要將圖片中的物體 label (標示) 起來讓電腦去學習，
  - 此類的labeling方式是Bounding Box Annotation。
  - 例：Car (10, 15. 680. 320)
- **物件辨識(Object detection)**：
  - 希望能看到圖片中所有的物體，因此，我們需要將圖片中所有希望辨識出的物體label起來，最常使用的方式也是Bounding Box Annotation。

影像辨識的種類及labeling

- 語意分割(Semantic Segmentation)：
  - 識別出圖像中的物件，需能明確劃分其範圍，此類的labeling並不僅僅拉四邊形方框，而是要在物體邊緣點出各點來標示物體範圍，方式有pixel-level 或 Polygonal annotation。



- 實例分割(Instance Segmentation)：
  - 除了要識別出物件類別，還需要區分出該類中各個物件，使用的label方式也是pixel-level 或Polygonal annotation。

Labeling工作的進行方式

- 自行訓練 Internal labeling：
- 外包Outsourcing或眾包Crowsourcing：
- 專業的labeling公司：
  - CloudFactory、Mighty AI、LQA、CrowdFlower、CapeStart以及DataPure
- 半自動Synthetic labeling：
  - Generative Adversarial Networks (GANs) 藉由GAN的方式來產生擬真的圖形與資料、再自動標識label。
  - Autoregressive models (ARs) Autoregressive model簡稱AR模型，利用線性迴歸依據舊有資料（從x0導出y0）來推估未來（用x0來預測x1，又稱為自迴歸）。多應用於時間序列類的模型，如經濟學、資訊學、自然現象的預測等。
  - Variational Autoencoders (VAEs) Autoencoder原本用於資料的降維或特徵的抽取，但現在也經常用於生成模型。

Dataset labeling tools

- **Tool: web based以及desktop based**
- web based
  - 跨平台且支援多人協同使用和進度管理，缺點是速度及後端設備需求較多（額外的web server 及 database server）。
- desktop based
  - 即裝即用，單機個人使用方便，缺點是非跨平台、多人管理不便…等。
- 某些完整的labeling tool可同時支援圖片、影片、聲音、矩形框、多邊框、像素點的標識，以及支援不同的標識格式，但是這類絕大多數為付費軟體，如RectLabel。
- FastAnnotationTool（https://github.com/christopher5106/FastAnnotationTool）
  - 使用OpenCV影像處理的協助，只要選取兩點便能快速的label物件。
- VoTT:（https://github.com/Microsoft/VoTT）
  - 全名是Visual Object Tagging Tool，微軟。
- LabelMe（https://github.com/wkentaro/labelme）

Yolo_mark 教學

https://github.com/AlexeyAB/Yolo_mark

標記軟體 Yolo_mark

- https://github.com/AlexeyAB/Yolo_mark
- **Windows** & **Linux** GUI for marking bounded boxes of objects in images for training Yolo v3 and v2
- Linux: 安裝
  - $ **git clone** https://github.com/AlexeyAB/Yolo_mark.git
  - **$ cd Yolo_mark**
  - **$ cmake .**
  - **$ make**
  - **$ linux_mark.sh**

- 使用教學
  - https://github.com/AlexeyAB/Yolo_mark
  - 中文 https://www.itread01.com/content/1547413031.html

Yolo_mark for Windows

- Download files:
  - **$ git clone git@github.com:AlexeyAB/Yolo_mark.git**
- To compile on **Windows** open yolo_mark.sln in MSVS2013/2015, compile it x64 & Release and **run the file: x64/Release/yolo_mark.cmd**. Change paths in **yolo_mark.sln** to the OpenCV 2.x/3.x installed on your computer:
  - (right click on project) -> properties -> C/C++ -> General -> Additional Include Directories: C:\opencv_3.0\opencv\build\include;
  - (right click on project) -> properties -> Linker -> General -> Additional Library Directories: C:\opencv_3.0\opencv\build\x64\vc14\lib;

Yolo_mark 執行與設定

- To test, simply run
  - on Windows: **x64/Release/yolo_mark.cmd**
  - on Linux: $ **./linux_mark.sh**
- To use for labeling your custom images:
  - **delete all files** from directory **x64/Release/data/img**
  - put your **.jpg-images** to this directory **x64/Release/data/img**
  - **(訓練影像檔，裡面包含圖片和每一張圖片對應的標注資料。)**
  - change number of classes (objects for detection) in file **x64/Release/data/obj.data**:

```
classes= 2
train = data/train.txt
valid = data/valid.txt
names = data/obj.names
backup = backup/
```

執行與設定

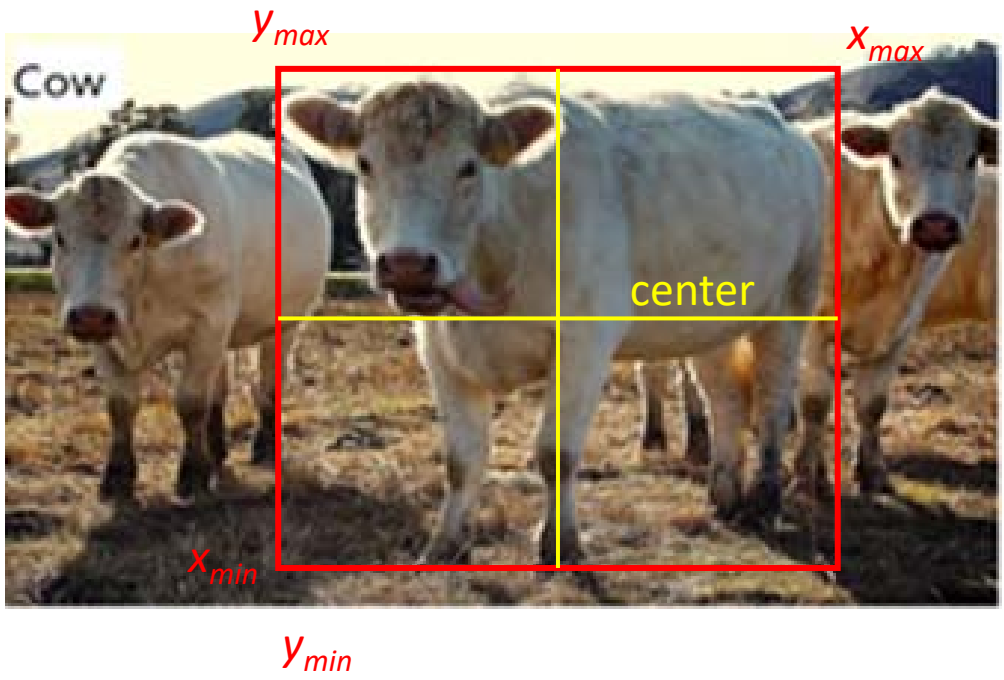
- put names of objects, one for **each line** in file **x64/Release/data/obj.names**: (資料集所有分類類名)
- 範例 (兩類)
  - air
  - bird
- **x64/Release/data/train.txt**是資料集的**相對路徑**集合。

data/img/0.jpg
data/img/1.jpg
data/img/2.jpg
data/img/3.jpg
data/img/4.jpg
data/img/5.jpg
data/img/6.jpg
data/img/7.jpg

- run file: **x64\Release\yolo_mark.cmd** or **$ ./linux_mark.sh to mark**
- 將所有類別以object id的形式展示出來

準備資料集

- 準備資料集，label bounding Box 的格式要轉為 txt 檔
- Yolo 的格式，它是由class id, 歸一化後的 x, y 中心座標及歸一化後的w, h 所組成，資料會呈現下圖的樣子



$$x_{center} = \frac{(x_{min} + x_{max})}{2}\frac{1}{w}$$

$$y_{center} = \frac{(y_{min} + y_{max})}{2}\frac{1}{h}$$

$$yolo_w = \frac{(x_{max} - x_{min})}{w}$$

$$yolo_h = \frac{(y_{max} - y_{min})}{h}$$

Yolo_mark

1. Select Image no
2. Select Object id
3. Mark bounding box
   left: draw box
   Right: move box

- 1 第一個數位代表類別，後面的四個資料是標註框的資訊，分別是標註框中心點的 x，y座標，標註框的寬w，高h。
- 每一行是一個標註物件，如果一張圖片中有多個目標，則txt就對應有多少行。
- 0 0.341797 0.547917 0.049219 0.118056
- 0 0.731250 0.581944 0.050000 0.225000

After mark

名称	日期	类型	大小	标记
0	2018/4/19 16:53	看图王 JPG 图片文...	39 KB	
0	2018/4/19 16:57	文本文档	1 KB	
1	2018/4/19 16:53	看图王 JPG 图片文...	40 KB	
1	2018/4/19 16:57	文本文档	1 KB	
2	2018/4/19 16:53	看图王 JPG 图片文...	40 KB	
2	2018/4/19 16:57	文本文档	1 KB	
3	2018/4/19 16:53	看图王 JPG 图片文...	40 KB	
3	2018/4/19 16:57	文本文档	1 KB	
4	2018/4/19 16:53	看图王 JPG 图片文...	40 KB	
4	2018/4/19 16:57	文本文档	1 KB	
5	2018/4/19 16:53	看图王 JPG 图片文...	41 KB	
5	2018/4/19 16:57	文本文档	1 KB	
6	2018/4/19 16:53	看图王 JPG 图片文...	39 KB	
6	2018/4/19 16:57	文本文档	1 KB	
7	2018/4/19 16:53	看图王 JPG 图片文...	40 KB	
7	2018/4/19 16:57	文本文档	1 KB	

Keyboard Shortcuts

Shortcut	Description
→	Next image
←	Previous image
r	Delete selected box (mouse hovered)
c	Clear all marks on the current image
p	Copy previous mark
o	Track objects
ESC	Close application
n	One object per image
0-9	Object id
m	Show coords
w	Line width
k	Hide object name
h	Help

LabelImg

準備訓練影像 LabelImg (method 1)

- 影像標記 (annotation) 會以 XML (PASCAL VOC format) 格式儲存。
- 在Windows/Linux 環境準備影像軟體 LabelImg.exe v.1.8.0
- Download link https://tzutalin.github.io/labelImg/
- Demo video:
  - https://www.youtube.com/watch?v=p0nR2YsCY_U&feature=youtu.be
- 安裝指令 (Linux) v1.4.3
  - **$ sudo apt-get install pyqt5-dev-tools**
  - **$ sudo pip3 install -r requirements/requirements-linux-python3.txt**
  - **$ make qt5py3**
  - **$ python3 labelImg.py**
  - **$ python3 labelImg.py [IMAGE_PATH] [PRE-DEFINED CLASS FILE]**

安裝 lableImg (method 2)

- 網站 https://github.com/tzutalin/labelImg
  - $ git clone https://github.com/tzutalin/labelImg
  - $ sudo apt-get install pyqt5-dev-tools
  - $ cd labelImg
  - $ sudo pip3 install -r requirements/requirements-linux-python3.txt (出現error)
  - $ make qt5py3
  - $ pyrcc5 -o resources.py resources.qrc
- 2- Change the location of resources.qrc and resource files to libs folder.
- 3- typing in cmd:
  - $ cd labelImg/libs
  - $ pyrcc5 -o resources.py resources.qrc
  - $ cd ..
- 4 執行
  - $ python3 labelImg.py
  - $ python3 labelImg.py [IMAGE_PATH] [PRE-DEFINED CLASS FILE]

參考 https://github.com/tzutalin/labelImg/issues/404

LabelImg 使用教學

- 中文使用教學：
  - Windows版路徑不可用中文
  - Username 中文無打開
    - #無法開啟labelimg或者閃退嗎？
      刪除C:\Users\yourUserName\當中的.labelImgSettings.pkl檔案，再重新開labelimg.exe
  - https://blog.gtwang.org/useful-tools/labelimg-graphical-image-annotation-tool-tutorial/

- 下載安裝 https://tzutalin.github.io/labelImg/
  win:1.8.0/linux:1.4.3
- 另外也可以從 PyPI 上面直接下載 (may be error) ：
  - **$ sudo pip3 install labelImg**
  - **$ labelImg**

LabelImg 設定

- For Yolo
  - In **data/predefined_classes.txt** define the list of classes that will be used for your training.
  - Build and launch using the instructions above.
  - Right below "**Save**" button in the toolbar, click "**PascalVOC**" button to switch to YOLO format.
  - You may use Open/OpenDIR to process single or multiple images.
  - **When finished with a single image, click save**.
- 中文教學網站
  - https://blog.gtwang.org/useful-tools/labelimg-graphical-image-annotation-tool-tutorial/

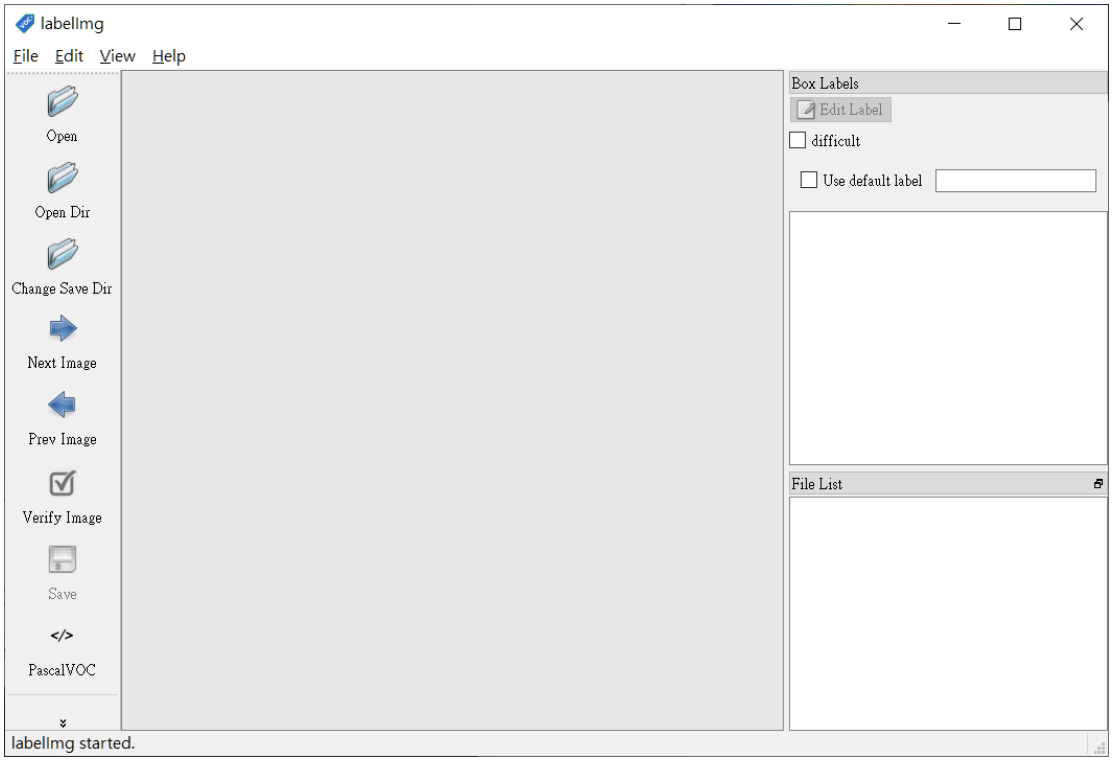
https://blog.gtwang.org/useful-tools/labelimg-graphical-image-annotation-tool-tutorial/

建議

- 下載 LabelImg (Windows版本)，Windows 新版有當機問題，本文使用 Windows_v1.8.0 版本。
- https://tzutalin.github.io/labelImg/

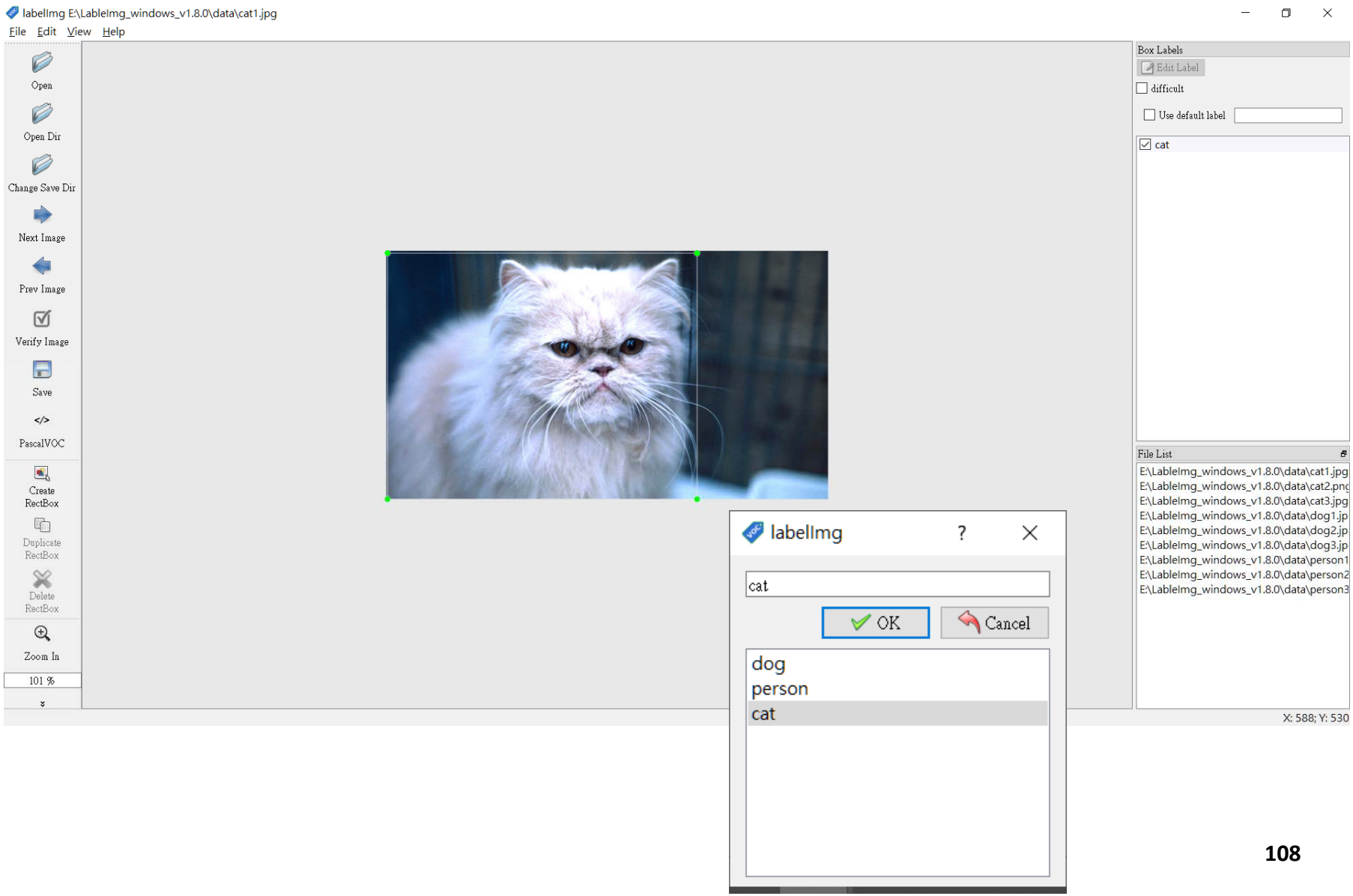
- Linux_v1.4.0
- Windows_v1.4.3
- Linux_v1.4.3
- Windows_v1.5.0
- Windows_v1.5.1
- Windows_v1.5.2
- Windows_v1.6.0
- Windows_v1.6.1
- Windows_v1.7.0
- Windows_v1.8.0

- 1. 解壓縮下載的檔案，開啟與labelImg.exe
- 2. 同層資料夾下data/predefined_classes.txt (建議使用 Notepad++ 開啟)，編輯欲標記的類別名稱，下圖為3個類別。
  - Cat
  - dog
  - person
- 3. 開啟 labelImg.exe
- 4. 請**將欲標記的圖檔放在同一資料夾下**，點選左邊 Menu 選單中的 **Open Dir** (全部目錄)/**open** (單一檔案)，選擇放置圖檔的資料夾/檔案。

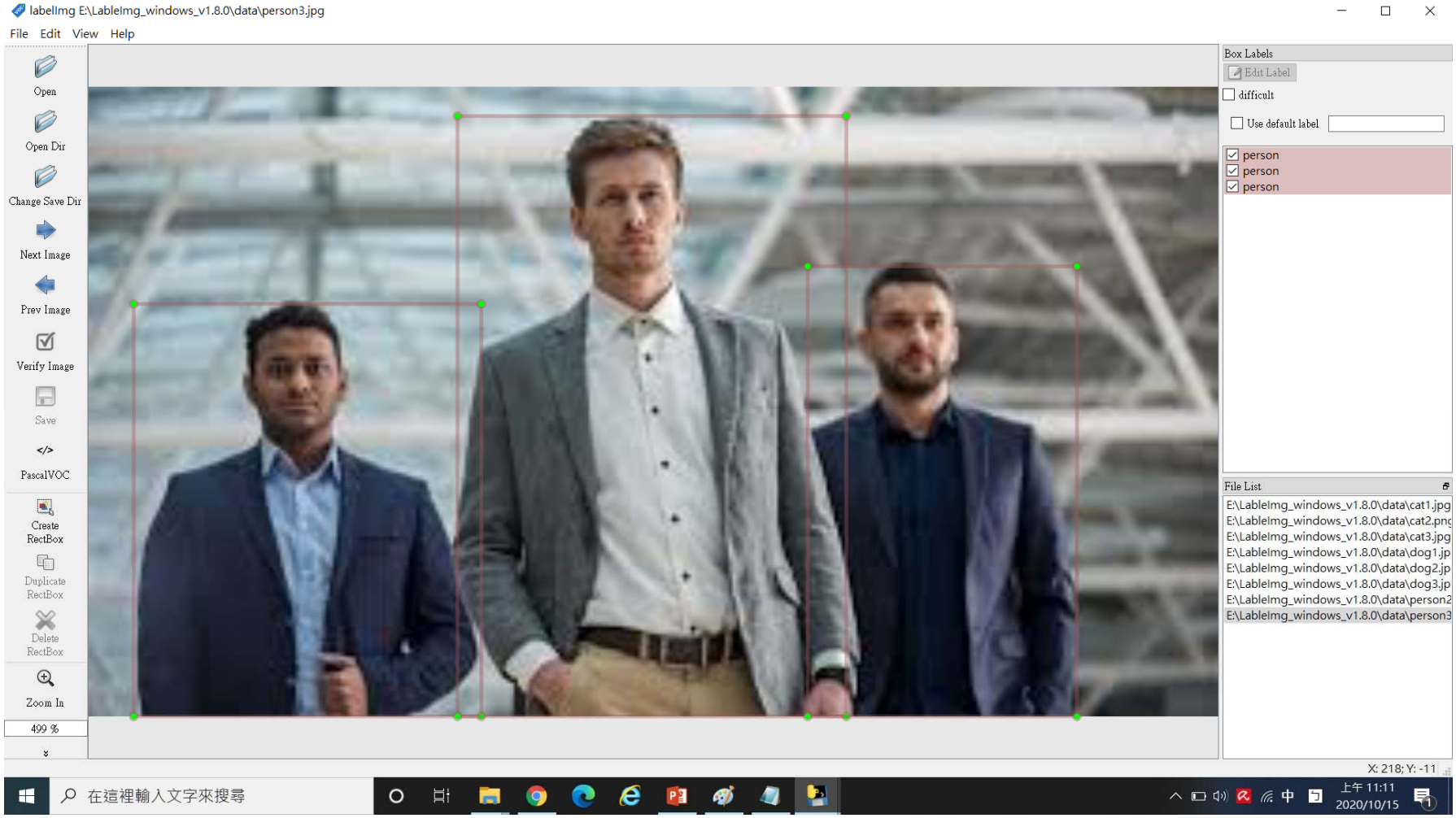


- 5. 點選左邊 Menu 選單中的 **Change Save Dir** ，設定 annotation 標記(XML 檔案)的儲存路徑。
- 6. 點選左邊Menu選單中的 **Create RectBox** (或按**mouse右鍵**)。
- 7. 在影像上用滑鼠拖曳的方式選取欲標記的物件，接著選擇此物件的類別。
- 8. 標記完後，按下左邊 Menu 選單中的 Save ，儲存標記。
- **每標一個物件或圖片要save (重要)**
- 9. 按下 Next Image ，標記下一張影像。
- 10. 完成所有影像的標記後，即可以在儲存標記的資料夾中得到 XML 檔案。

labelImg 畫面


labelImg E:\LabelImg_windows_v1.8.0\data\cat1.jpg
File Edit View Help
Open
Open Dir
Change Save Dir
Next Image
Prev Image
Verify Image
Save
PascalVOC
Create RectBox
Duplicate RectBox
Delete RectBox
Zoom In
101 %
Box Labels
Edit Label
difficult
Use default label
cat
File List
E:\LabelImg_windows_v1.8.0\data\cat1.jpg
E:\LabelImg_windows_v1.8.0\data\cat2.png
E:\LabelImg_windows_v1.8.0\data\cat3.jpg
E:\LabelImg_windows_v1.8.0\data\dog1.jp
E:\LabelImg_windows_v1.8.0\data\dog2.jp
E:\LabelImg_windows_v1.8.0\data\dog3.jp
E:\LabelImg_windows_v1.8.0\data\person1
E:\LabelImg_windows_v1.8.0\data\person2
E:\LabelImg_windows_v1.8.0\data\person3
X: 588; Y: 530
labelImg
cat
OK
Cancel
dog
person
cat

多物件範例


labelImg E:\LabelImg_windows_v1.8.0\data\person3.jpg
File Edit View Help
Open
Open Dir
Change Save Dir
Next Image
Prev Image
Verify Image
Save
PascalVOC
Create RectBox
Duplicate RectBox
Delete RectBox
Zoom In
499 %
Box Labels
Edit Label
difficult
Use default label
person
person
person
File List
E:\LabelImg_windows_v1.8.0\data\cat1.jpg
E:\LabelImg_windows_v1.8.0\data\cat2.png
E:\LabelImg_windows_v1.8.0\data\cat3.jpg
E:\LabelImg_windows_v1.8.0\data\dog1.jp
E:\LabelImg_windows_v1.8.0\data\dog2.jp
E:\LabelImg_windows_v1.8.0\data\dog3.jp
E:\LabelImg_windows_v1.8.0\data\person2
E:\LabelImg_windows_v1.8.0\data\person3
X: 218; Y: -11
在這裡輸入文字來搜尋
上午 11:11
2020/10/15

Hotkeys

Hotkeys

Ctrl + u	Load all of the images from a directory
Ctrl + r	Change the default annotation target dir
Ctrl + s	Save
Ctrl + d	Copy the current label and rect box
Space	Flag the current image as verified
w	Create a rect box
d	Next image
a	Previous image
del	Delete the selected rect box
Ctrl++	Zoom in
Ctrl--	Zoom out
↑→↓←	Keyboard arrows to move selected rect box

快捷鍵

Ctrl + u	載入資料夾中全部的圖片
Ctrl + r	更改標籤檔的預設路徑
Ctrl + s	儲存
Ctrl + d	複製當前的 RectBox 與標籤
Space	將目前的照片設為已驗證
w	創建一個 RectBox
d	切換至下一張照片
a	切換回上一張照片
del	刪除目前選取的RectBox
Ctrl++	放大
Ctrl--	縮小
↑→↓←	可用方向鍵微調 RectBox 的位子

https://jason-chen-1992.weebly.com/home/-labelimg

訓練 YOLOv4

YOLOV4訓練資料目錄結構範例

- 建議
  - Project name: deron1
  - 所有檔案可以打包移機處理deron1.zip
  - 建立目錄
    - DIR **data/obj** //for all training images and .txt files
    - DIR **data/val** //for all validate images and .txt files
    - DIR **weights** //for storing final weights
    - File **deron1.cfg** //for training yolov4 cfg
    - File **deron1.data**
    - File **deron1.names**
    - File **deron1_test.cfg** //for testing yolov4 cf
    - File **deron1_train.txt**
    - File **deron1_val.txt**

訓練yolov4 步驟 (官網)

- **Step 0:**
  - For training **deron1.cfg** download the **pre-trained weights-file** (162 MB): yolov4.conv.137
  - **$ wget https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.conv.137**

- **Step 1:**
  - Create file **deron1.cfg** with the same content as in **yolov4-custom.cfg** (or copy yolov4-custom.cfg to **yolo-obj.cfg**) and:
  - change line batch to batch=64
  - *# GPU memory不夠的圖像可以改為16,32等*
  - change line subdivisions to subdivisions=16 **#可以再上修64**

參數計算

- yolov4 偵測的濾鏡(filter) 大小為 (C+5)*B
  - - B 是每個Feature Map可以偵測的bndBox數量，這裡設定為3
  - - 5 是bndBox輸出的5個預測值: x, y, w ,h 以及 Confidence
  - - C 是類別數量
  - **classes=3** #類別數 cat, dog, person
  - **filters=(classes + 5)*3** # 因為是3類別，所以filters更改為 **24**
  - #在yolo layer 前的convent. Layer 共三個位置
- Sed 指令 **sed OPTIONS... [SCRIPT] [INPUTFILE...]**
  - OPTION - 選項
  - SCRIPT - 動作草稿
  - INPUTFILE - 檔案輸入
  - 參考 https://medium.com/@chingi071/yolo-c49f70241aa7

如何修改cfg網路結構 deron1.cfg

- *# Testing*
- batch=64 *# GPU memory不夠的圖像可以改為16,32等*
- subdivisions=16 *#這裡需要改 64 除非有超強GPU與記憶體*

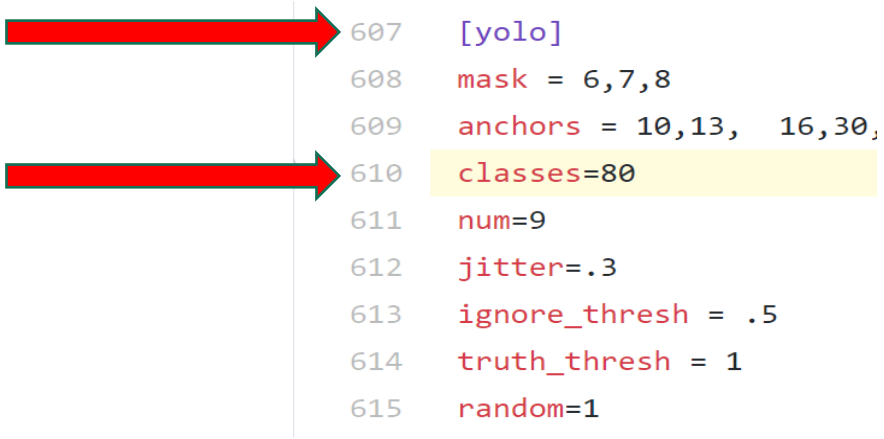
- *#Training*
- width=416 *#可以改為32的倍數*
- height=416 *#可以改為32的倍數*
- channels=3
- momentum=0.949
- decay=0.0005
- angle=0
- saturation = 1.5
- exposure = 1.5
- hue=.1

如何修改cfg網路結構 deron1.cfg

- **[net]開頭這部分**
  - learning_rate=0.001
  - burn_in=1000
  - **max_batches = 4000** *#改為classses*2000，但不小於6000* (or 4000)
  - policy=steps
  - **steps=3200,3600** *#改為max_batch的0.8倍和0.9倍*
  - scales=.1,.1
  - *#cutmix=1*
  - mosaic=1
  - 你需要按照你的**類別數目**，修改這個max_batches, 通常就是2000x類別數。
  - 然後steps就是上面數字的80%和90%的分界點。

Step 1

- change line max_batches to (classes*2000 but not less than number of training images, but not less than number of training images and not less than 6000), f.e. max_batches=6000 if you train for 3 classes
- change line steps to 80% and 90% of max_batches, f.e. steps=4800,5400
- set network size width=416 height=416 or any value multiple of 32: #太小辨識率不佳
- change line classes=3 to your number of objects **in each of 3 [yolo]-layers**



Step 1

- change **[filters=255]** to **filters=(classes + 5)x3** in the **3 [convolutional] before each [yolo] layer**, keep in mind that **it only has to be the last [convolutional] before each of the [yolo] layers**.
- when using **[Gaussian_yolo]** layers, change **[filters=57] filters=(classes + 9)x3** in the 3 [convolutional] before each [Gaussian_yolo] layer
- So
  - if **classes=1** then should be filters=18.
  - If **classes=2** then write filters=21.
  - (Do not write in the cfg-file: filters=(classes + 5)x3)

```
[convolutional]
filters=21

[region]
classes=2
```

- (Generally filters depends on the classes, coords and number of masks, i.e. filters=(classes + coords + 1)*<number of mask>, where mask is indices of anchors. If mask is absence, then filters=(classes + coords + 1)*num)

Step 2&3

- Step 2:
  - Create file deron1.names in the directory **~/darknet/deron1/**, with objects names - each in new line
    - Dog
    - Person
    - cat

- Step 3:
  - Create file **deron1.data** in the directory **~/darknet/deron1/,** containing (where classes = number of objects):
  - #若找不到用絕對路徑
  - **classes = 3**
  - **train = deron1/deron1_train.txt**
  - **valid = deron1/deron1_val.txt**
  - **names = deron1/deron1.names**
  - **backup = deron1/weights**

Steps 4&5

- **Step4:**
  - Put image-files (.jpg) of your objects in the directory **~/darknet/deron1/data/obj #training data**
  - **~/darknet/deron1/data/val #val. Data**

- **Step 5:**
  - You **should label** each object on images from your dataset. Using YOLO_mark or labelImg
  - It will create .txt-file for each .jpg-image-file - in the same directory and with the same name, but with .txt-extension,
  - and put to file: object number and object coordinates on this image, for each object in new line:
  - **<object-class> <x_center> <y_center> <width> <height>**

Step 5

- Where:
  - <object-class> - integer object number from 0 to (classes-1)
  - <x_center> <y_center> <width> <height> - float values relative to width and height of image, it can be equal from (0.0 to 1.0]
- **For example**:
  - <x> = <absolute_x> / <image_width> or
  - <height> = <absolute_height> / <image_height>
  - <x_center> <y_center> - are center of rectangle (are not top-left corner)
- **For example** for img1.jpg you will be created img1.txt containing:
  - 1 0.716797 0.395833 0.216406 0.147222
  - 0 0.687109 0.379167 0.255469 0.158333
  - 1 0.420312 0.395833 0.140625 0.166667

Step 6

- Step 6:
  - Create file deron1_train.txt in directory **~/darknet/deron1/,** with filenames of your images, each filename in new line, with path relative to darknet.exe, for example containing:

  - #若找不到用絕對路徑
    - ./data/obj/img1.jpg
    - ./data/obj/img2.jpg
    - ./data/obj/img3.jpg

Step 7

- Download pre-trained weights for the convolutional layers and put to the directory ~\darknet\
  - for **yolov4.cfg**, yolov4-custom.cfg (162 MB): yolov4.conv.137 (Google drive mirror yolov4.conv.137)
  - for yolov4-tiny.cfg, yolov4-tiny-3l.cfg, yolov4-tiny-custom.cfg (19 MB): yolov4-tiny.conv.29
  - for **csresnext50-panet-spp.cfg** (133 MB): csresnext50-panet-spp.conv.112
  - **for yolov3.cfg**, yolov3-spp.cfg (154 MB): darknet53.conv.74
  - for **yolov3-tiny-prn.cfg** , yolov3-tiny.cfg (6 MB): yolov3-tiny.conv.11
  - for **enet-coco.cfg** (EfficientNetB0-Yolov3) (14 MB): enetb0-coco.conv.132

Steps 8&9 開始訓練訓練指令

- Step 8:
  - **$ ./darknet detector train deron1/deron1.data deron1.cfg yolov4.conv.137**

◦ (file `yolo-obj_last.weights` will be saved to the `build\darknet\x64\backup\` for each 100 iterations)

◦ (file `yolo-obj_xxxx.weights` will be saved to the `build\darknet\x64\backup\` for each 1000 iterations)

◦ (to disable Loss-Window use `darknet.exe detector train data/obj.data yolo-obj.cfg yolov4.conv.137 -dont_show`, if you train on computer without monitor like a cloud Amazon EC2)

◦ (to see the mAP & Loss-chart during training on remote server without GUI, use command `darknet.exe detector train data/obj.data yolo-obj.cfg yolov4.conv.137 -dont_show -mjpeg_port 8090 -map` then open URL `http://ip-address:8090` in Chrome/Firefox browser)

- Step 9:
  - After training is complete - get result deron1_final.weights from path ~/darknet/deron1/weights
  - After each **100 iterations** you can stop and later start training from this point.
  - Also you can get result earlier than all 45000 iterations.

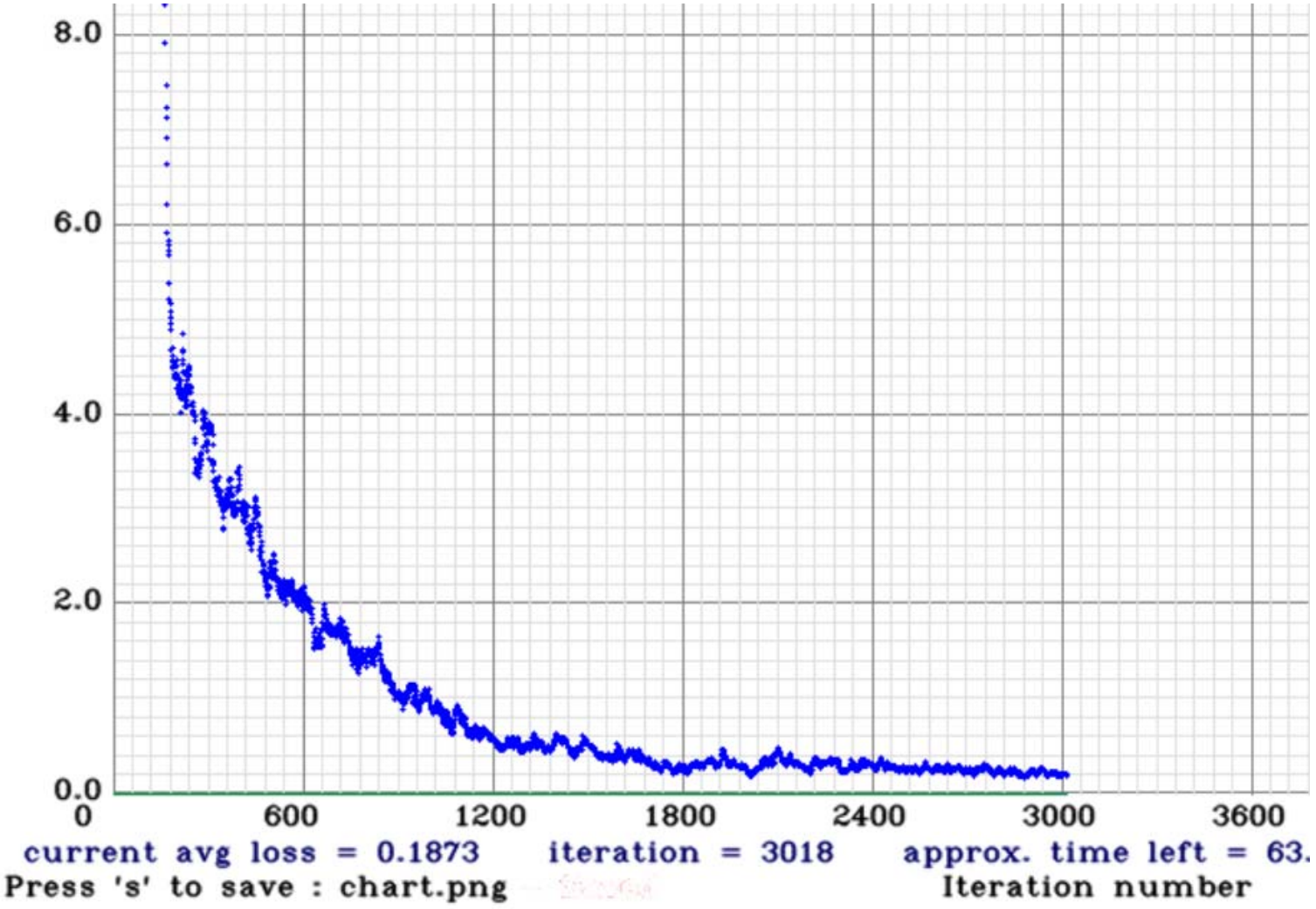
訓練自己的yolov4模型

- 訓練的command，
- **Format**
  - **$ ./darknet detector train {obj.data檔的path} {yolov4.cfg檔的path} {pre-trained weights的path} -dont_show -mjpeg_port 8090 -gpus 0,1**
- **Train Example**
  - **$ ./darknet detector train deron1/deron1.data deron1/deron1.cfg** yolov4.conv.137 **-gpus 0**
- **Test Example**
  - **$ ./darknet detector test deron1/deron1.data deron1/deron1.cfg deron1/weights/deron1_last.weights –map**

訓練指令

- ./darknet detector train {obj.data檔的path} {yolov4.cfg檔的path} {pre-trained weights的path} -dont_show -mjpeg_port **8090 -gpus 0,1**
- **參數說明：**
  - {obj.data檔的path}：即yolov4_config path下的obj.data
  - {yolov4.cfg檔的path}：即yolov4_config path下的yolov4.cfg
  - {pre-trained weights的path}：從YOLOV4官方網站下載預訓練檔path，或從上次最後訓練的weights檔path。
  - dont_show：傳承自YOLO-Alexey優化版的功能，若不加上此參數，則會顯示training過程的loss變化曲線圖，但如果你是遠端ssh連線訓練，則可加上此參數不顯示。
  - -mjpeg_port 8090：傳承自YOLO-Alexey優化版的功能，可於訓練主機的8090顯示訓練曲線圖表。
  - -gpus 0,1：指定要在那些GPU training。

訓練曲線圖


8.0
6.0
4.0
2.0
0.0
0
600
1200
1800
2400
3000
3600
current avg loss = 0.1873 iteration = 3018 approx. time left = 63.
Press 's' to save : chart.png
Iteration number

Select which model is suitable for training on your GPU

- for 4GB-RAM GPUs
  - YOLOv4-Leaky-416: batch=64, subdivisions=64.

- for 6GB-RAM GPUs
  - YOLOv4-Leaky-416: batch=64, subdivisions=32.
  - YOLOv4-Mish-416: batch=64, subdivisions=64.

- for 8GB-RAM GPUs
  - YOLOv4-Mish-416: batch=64, subdivisions=32.

- for 11GB-RAM GPUs (1080ti, 2080ti)
  - YOLOv4-Leaky-416: batch=64, subdivisions=16.
  - YOLOv4-Mish-416: batch=64, subdivisions=16.

相關連結

- 中文說明
  - https://wings890109.pixnet.net/blog/post/68939643-yolo-v4-%E8%BE%A8%E8%AD%98%E8%87%AA%E5%AE%9A%E7%BE%A9%E7%89%A9%E4%BB%B6
- 簡體詳細教程
  - https://my.oschina.net/u/4321806/blog/4470154
- Python 訓練教程
  - https://www.pythonf.cn/read/100217
- 影片連結
  - Yolo V4 - How it Works & Why it's So Amazing!
    - https://www.youtube.com/watch?v=_JzOFWx1vZg
  - YOLOv4 Tutorial #1 - Prerequisites for YOLOv4 Installation in 10 Steps
    - https://www.youtube.com/watch?v=5pYh1rFnNZs
  - YOLOv4 Tutorial #2 - How to Run YOLOv4 on Images and video
    - https://www.youtube.com/watch?v=sUxAVpzZ8hU
  - YOLOv4 Tutorial #3 - YOLOv4 on Webcam & Python Code Explanation
    - https://www.youtube.com/watch?v=R-ZutgHnAhQ

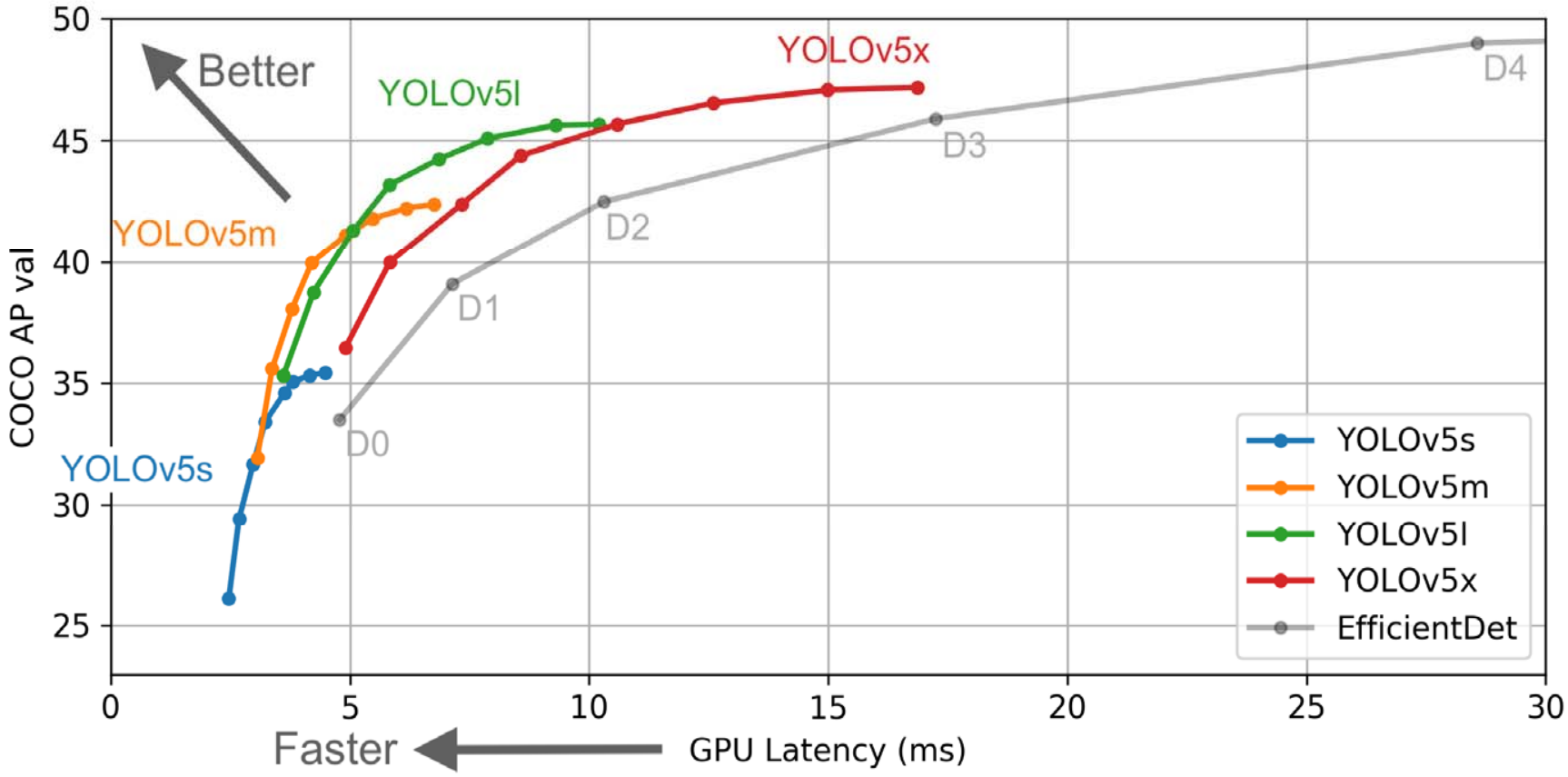
YOLO V5

Yolov5

- YOLOV4發表未滿兩個月，便突然出現了打著YOLOV5名號的新模型，號稱較前代改進的幅度更大。
- YOLOV5包含了一系列從小到大型的5s, 5m, 5l, 5x模型，
- 而與YOLOV4相對的是最小最弱的5s，但5s不但檢測速度更快更準，模型大小還縮小至只有十分之一，甚至還小於YOLOV3-Tiny。
- 可能由於原作者Roboflow也是與YOLO淵源頗深，先前所開發的YOLO Pytorch版大受歡迎之故，否則，這麼優秀的模型另外起個新名字也應能大受矚目。
- 網站
  - https://github.com/ultralytics/yolov5

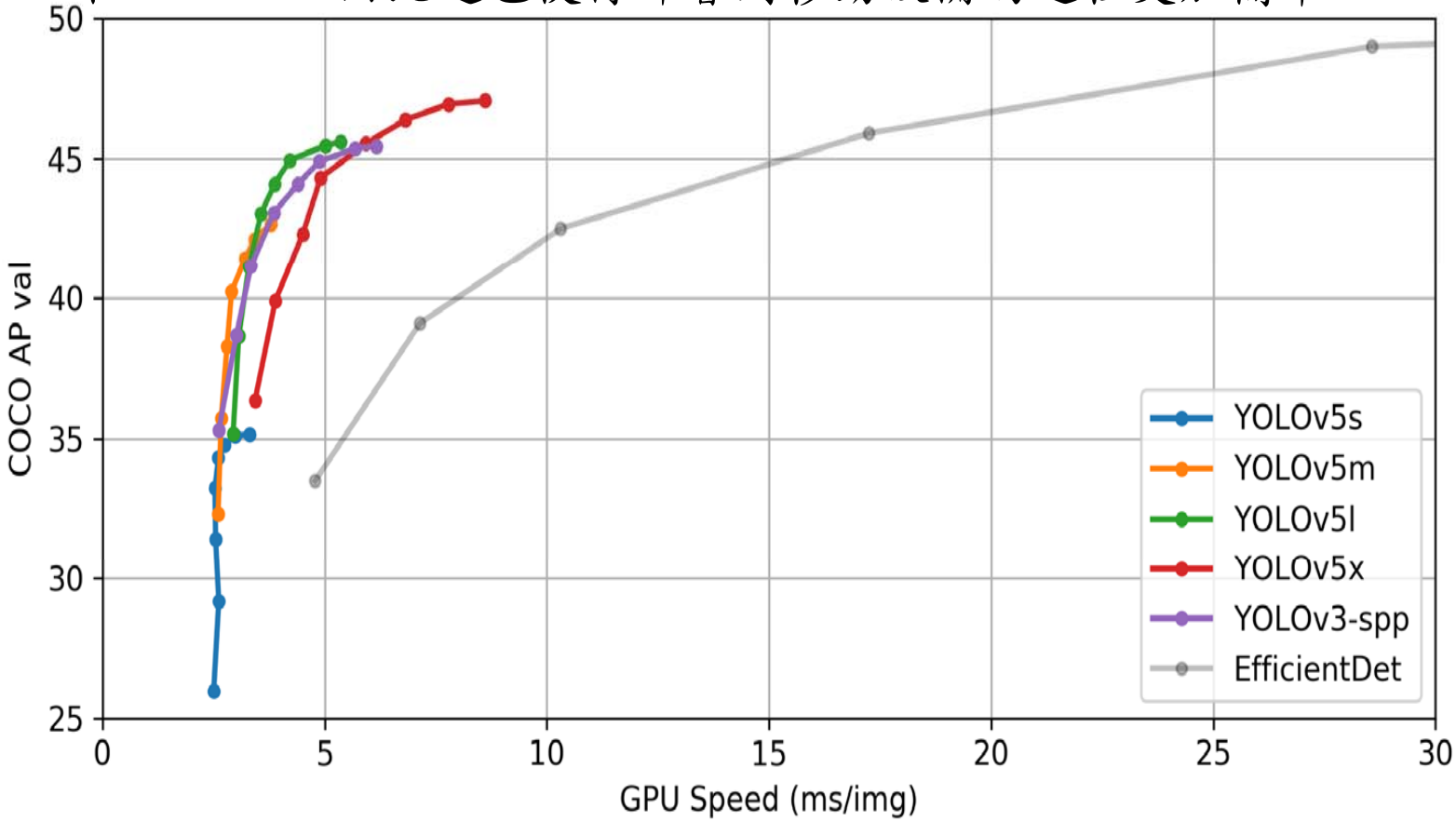
Related web

- YOLOV5 9 June 2020
  - https://github.com/ultralytics/yolov5



比較

- YOLOv5 s的權重文件為**27MB**。YOLOv4的權重文件為**244MB**。YOLOv5比YOLOv4小近90%。這意味著YOLOv5可以部署到**嵌入式設備**。
- 此外，因為YOLOv5是在PyTorch中實現的；YOLOv5還可以編譯為ONNX和CoreML，因此這也使得部署到移動設備的過程更加簡單。



Different models s/m/l/x

Pretrained Checkpoints

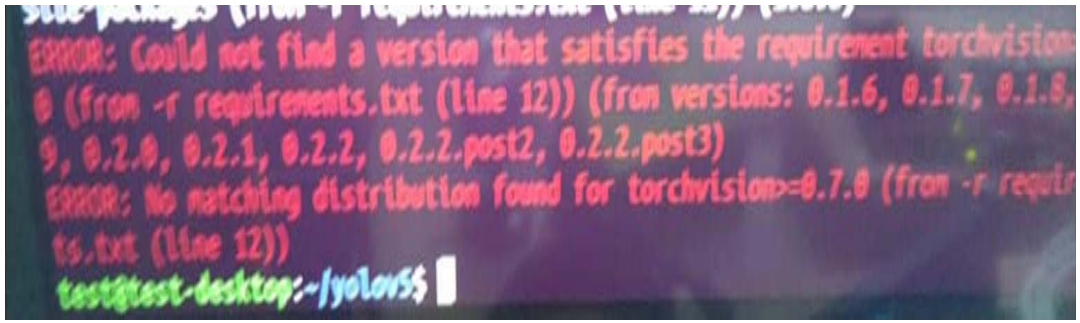
Model	AP^{val}	AP^{test}	AP_{50}	$Speed_{GPU}$	FPS_{GPU}		params	FLOPS
YOLOv5s	37.0	37.0	56.2	**2.4ms**	**416**		7.5M	13.2B
YOLOv5m	44.3	44.3	63.2	3.4ms	294		21.8M	39.4B
YOLOv5l	47.7	47.7	66.5	4.4ms	227		47.8M	88.1B
YOLOv5x	**49.2**	**49.2**	**67.7**	6.9ms	145		89.0M	166.4B
YOLOv5x + TTA	**50.8**	**50.8**	**68.9**	25.5ms	39		89.0M	354.3B
YOLOv3-SPP	45.6	45.5	65.2	4.5ms	222		63.0M	118.0B

YOLOv5 修正版

- YOLOV5 9 June 2020
  - https://github.com/ultralytics/yolov5
- V 2.0 July 23, 2020
- V 3.0 Augest 13, 2020
  - https://github.com/ultralytics/yolov5/releases
- Code download
  - https://github.com/ultralytics/yolov5/archive/v3.0.zip
- 簡體文件
  - https://zhuanlan.zhihu.com/p/161778675

Environment Setup

- Will require PyTorch version ≥ 1.7, Python version>= 3.6, and CUDA version 10.2. (used *ubuntu 16.04)*
- **Install**
  - **$ git clone https://github.com/ultralytics/yolov5**
  - **$ cd yolov5**
  - **$ pip install -r requirements.txt**
- **requirements.txt**
  - **#$ pip3 install -r requirements.txt**
  - Cython
  - matplotlib>=3.2.2
  - numpy>=1.18.5
  - opencv-python>=4.1.2
  - Pillow PyYAML>=5.3
  - scipy>=1.4.1 tensorboard>=2.2 torch>=1.6.0 torchvision>=0.7.0
  - tqdm>=4.41.0



更新版requirement.txt

```
# pip install -r requirements.txt

# Base ----------------------------------------
matplotlib>=3.2.2
numpy>=1.18.5
opencv-python>=4.1.2
Pillow>=7.1.2
PyYAML>=5.3.1
requests>=2.23.0
scipy>=1.4.1
torch>=1.7.0
torchvision>=0.8.1
tqdm>=4.41.0

# Logging -------------------------------------
tensorboard>=2.4.1
# wandb

# Plotting ------------------------------------
pandas>=1.1.4
seaborn>=0.11.0

# Export --------------------------------------
# coremltools>=4.1  # CoreML export
# onnx>=1.9.0  # ONNX export
# onnx-simplifier>=0.3.6  # ONNX simplifier
# scikit-learn==0.19.2  # CoreML quantization
# tensorflow>=2.4.1  # TFLite export
# tensorflowjs>=3.9.0  # TF.js export

# Extras --------------------------------------
# albumentations>=1.0.3
# Cython  # for pycocotools
https://github.com/cocodataset/cocoapi/issues/172
# pycocotools>=2.0  # COCO mAP
# roboflow
thop  # FLOPs computation
```

Installing the YOLOv5 Environment

- To start off with YOLOv5 we first clone the YOLOv5 repository and install dependencies. This will set up our programming environment to be ready to running object detection training and inference commands.
  - **$ git clone https://github.com/ultralytics/yolov5 *# clone repo***
  - **$ cd yolov5**
  - **$ pip3 install -r requirements.txt *# install dependencies***
- On Google Colab
  - **import** torch
  - **from** IPython.display **import** Image *# for displaying images*
  - **from** utils.google_utils **import** gdrive_download *# for downloading models/datasets*
  - print('torch %s %s' % (torch.__version__, torch.cuda.get_device_properties(0) **if** torch.cuda.is_available() **else** 'CPU'))
- Output:
  - **torch** 1.5.0+cu101 _CudaDeviceProperties(name='Tesla P100-PCIE-16GB', major=6, minor=0, total_memory=16280MB, multi_processor_count=56)

細部安裝

- # install yolov5 v3.0 memo
- #download yolov5 from https://github.com/ultralytics/yolov5/releases/tag/v3.0
- unzip yolov5.3.0.
- $ **cd yolov5**
- $ **pip3 install -r requirements.txt**
- # it may generate some error
- # **install Pytorch v1.6.0** (可以考慮更高版本，安裝檔案要更新)
- #reference link: https://forums.developer.nvidia.com/t/pytorch-for-jetson-version-1-6-0-now-available/72048
- $ **wget https://nvidia.box.com/shared/static/ncgzus5o23uck9i5oth2n8n06k340l6k.whl -O torch-1.6.0-cp36-cp36m-linux_aarch64.whl**
- $ **sudo apt-get install python3-pip libopenblas-base libopenmpi-dev**
- $ **pip3 install Cython**
- $ **pip3 install numpy torch-1.6.0-cp36-cp36m-linux_aarch64.whl**

細部安裝 注意搭配版本

- #install steuptools
- **$ sudo apt-get install -y python-setuptools**
- #install torchvison
- **$ sudo apt-get install libjpeg-dev zlib1g-dev**
- **$ git clone --branch v0.7.0 https://github.com/pytorch/vision torchvision #不同torch要不同torchvision**
- # see below for version of torchvision to download
- **$ cd torchvision**
- **$ export BUILD_VERSION=0.7.0**
- **$ sudo python3 setup.py install**
- # use python3 if installing for Python 3.6
- **$ cd ..**
- **$ cd yolov5**
- **$ pip3 install tqdm**

不同版本安裝文件
https://qengineering.eu/install-pytorch-on-jetson-nano.html

下載權重檔 測試

- **YOLO v5的權重檔案：**
  - YOLOv5s/YOLOv5m/YOLOv5l/YOLOv5x https://drive.google.com/drive/folders/1Drs_Aiu7xx6S-ix95f9kNsA6ueKRpN2J
  - 放到yolov5目錄
- 要檢測一些圖像，您可以簡單地將它們放入名為**inference/images**的資料夾中，然後根據自動運行推斷:
- 測試指令
  - **$ python3 detect.py --weights yolov5s.pt**
- 我指定希望使用- view-img 標誌查看輸出，並將輸出存儲在位置**inference/output**中。這將在這個位置創建一個.mp4文件。
  - $ python3 detect.py --weights yolov5s.pt --source **inference/videos/messi.mp4** --view-img --output inference/output
- $ **python3 detect.py --weights yolov5s.pt --img-size 640 --conf-thres 0.25 --source ./data/images/bus.jpg**

執行範例

```
test@test-desktop:~/yolov5$ python3 detect.py --weights yolov5s.pt --img-size 64
0 --conf-thres 0.25 --source horses.jpg
Namespace(agnostic_nms=False, augment=False, classes=None, conf_thres=0.25, devi
ce='', img_size=640, iou_thres=0.5, output='inference/output', save_txt=False, s
ource='horses.jpg', update=False, view_img=False, weights=['yolov5s.pt'])
Using CUDA device0 _CudaDeviceProperties(name='NVIDIA Tegra X1', total_memory=39
56MB)

Fusing layers...
Model Summary: 191 layers, 7.46816e+06 parameters, 0 gradients
image 1/1 /home/test/yolov5/horses.jpg: 448x640 5 horses, Done. (2.368s)
Results saved to inference/output
Done. (66.475s)
test@test-desktop:~/yolov5$
```

```
test@test-desktop:~/yolov5$ python3 detect.py --weights yolov5s.pt --img-size 64
0 --conf-thres 0.25 --source dog.jpg
Namespace(agnostic_nms=False, augment=False, classes=None, conf_thres=0.25, devi
ce='', img_size=640, iou_thres=0.5, output='inference/output', save_txt=False, s
ource='dog.jpg', update=False, view_img=False, weights=['yolov5s.pt'])
Using CUDA device0 _CudaDeviceProperties(name='NVIDIA Tegra X1', total_memory=39
56MB)

Fusing layers...
Model Summary: 191 layers, 7.46816e+06 parameters, 0 gradients
image 1/1 /home/test/yolov5/dog.jpg: 512x640 1 bicycles, 1 cars, 1 trucks, 1 dog
s, 1 potted plants, Done. (3.891s)
Results saved to inference/output
```

Inference with detect.py

▼Inference with detect.py

`detect.py` runs inference on a variety of sources, downloading models automatically from the latest YOLOv5 release and saving results to `runs/detect`.

```
$ python detect.py --source 0  # webcam
                            img.jpg  # image
                            vid.mp4  # video
                            path/  # directory
                            path/*.jpg  # glob
                            'https://youtu.be/Zgi9g1ksQHc'  # YouTube
                            'rtsp://example.com/media.mp4'  # RTSP, RTMP, HTTP stream
```

Inference

- Self trained weights: **best.pt**
- To run the model inference use the following command.

  $ python3 detect.py --source sample_img/ --weights weights/best.pt --conf 0.4

- Parameters
  - — source: input images directory or single image path or video path
  - — weights: trained model path
  - — conf: confidence threshold
- Elephant weights https://github.com/mihir135/yolov5/blob/master/weights_elephant/last.pt
- Code download: https://github.com/mihir135/yolov5

Inference

- Inference with YOLOv5 and PyTorch Hub. Models automatically download from the latest YOLOv5 release.

```
import torch

# Model
model = torch.hub.load('ultralytics/yolov5', 'yolov5s')
# or yolov5m, yolov5l, yolov5x, custom

# Images
img = 'https://ultralytics.com/images/zidane.jpg'
# or file, Path, PIL, OpenCV, numpy, list

# Inference
results = model(img)

# Results
results.print()
# or .show(), .save(), .crop(), .pandas(), etc.
```

實驗比較

- YOLOV5 於樹莓派上
  - https://chtseng.wordpress.com/2020/07/03/yolov5%E5%9C%A8%E6%A8%B9%E8%8E%93%E6%B4%BE%E4%B8%8A%E7%9A%84%E6%B8%AC%E8%A9%A6/

	頭	身體	頭	身體	頭	身體
YOLOV3-Tiny	133	104	133	126	163	136
YOLOV5s-640	146	154	109	191	104	196
YOLOV5s-960	152	148	117	183	123	177
YOLOV5m-640	143	157	105	195	106	194
YOLOV5m-960	151	149	115	185	130	170
YOLOV5l-640	142	158	107	193	103	197

Yolov5 訓練

簡體文件

https://github.com/DataXujing/YOLO-v5

Web link https://towardsdatascience.com/yolo-v5-is-here-b668ce2a4908

YOLOv5 Tranning Steps

- **Training Steps**
  - Preparing Dataset
  - Environment Setup
  - Configure/modify files and directory structure
  - Training
  - Inference
  - Result
- Code download https://github.com/mihir135/yolov5

Preparing Dataset

- 所需格式與傳統YOLO相同，都是一個圖檔搭配一個txt標記檔，
- 但是**YOLOV5不用產生train.txt及test.txt**這兩個image list，只要將圖檔+txt檔的**路徑**設定給train及val這兩個參數即可 (紀錄於**yolov5s_deron1.yaml**)。
- 另外，它分離了**dataset定義**和**模型定義**成為兩個不同的設定檔案，不再像以往是單獨一個yolov4.cfg定義了dataset及model兩種設定，

YOLOv5 模型設定檔

- 從yolov5/models/下，選擇一個（本例為yolov5s.yaml）範本複製過來，修改。
- 注意，YOLOV5-s的建議尺寸是640x640非YOLOV4傳統的618x618 ，但依官方說明，該尺寸在訓練時可自行指定。
- 設定檔dataset.yaml (deron1.yaml)
  - data/coco128.yaml來自於COCO train2017資料集的前128個訓練圖像，可以基於該yaml修改自己資料集的yaml文件
- 內容
  - # train and val datasets (image directory or *.txt file with image paths)
  - **train: /home/deron/yolov5/deron1/deron1_train.txt**
  - **val: /home/deron/yolov5/deron1/deron1_val.txt**
  - **test: /home/deron/yolov5/deron1/deron1_val.txt**
  - **# number of classes**
  - **nc: 3**
  - **# class names**
  - **names: [‘cat', ‘person', ‘dog']**

2.創建標注文件

- 可以使用LabelImg, Labme, Labelbox, CVAT來標注資料，對於目標檢測而言需要標注bounding box即可。
- 與YOLOv4相容
- VOC XML 轉換函數
  - def convert(size, box):
  - '''
  - 將標注的xml檔標注轉換為darknet形的座標
  - '''
  - dw = 1./(size[0])
  - dh = 1./(size[1])
  - x = (box[0] + box[1])/2.0 - 1
  - y = (box[2] + box[3])/2.0 - 1
  - w = box[1] - box[0]
  - h = box[3] - box[2]
  - x = x*dw
  - w = w*dw
  - y = y*dh
  - h = h*dh
  - return (x,y,w,h)

3.組織訓練集的目錄

- 將訓練集train/驗證集val/測試集test的images和labels資料夾按照如下的方式進行存放
- (類似YOLOV4架構，避免修改太多)

4.選擇模型backbone進行模型設定檔的修改

- 在項目的./models資料夾下選擇一個需要訓練的模型(s/m/l/x)，這裡我們選擇yolov5x.yaml,最大的一個模型進行訓練，瞭解不同模型的大小和推斷速度。
- 如果你選定了一個模型，那麼需要修改模型對應的yaml文件(**yolov5s/m/l/x.yaml**)

```
# parameters
nc: 3  # number of classes   <------------------  UPDATE to match your d
depth_multiple: 1.33  # model depth multiple
width_multiple: 1.25  # layer channel multiple

# anchors
anchors:
  - [10,13, 16,30, 33,23]  # P3/8
  - [30,61, 62,45, 59,119]  # P4/16
  - [116,90, 156,198, 373,326]  # P5/32
```

Yolov5 backbone

```
# yolov5 backbone
backbone:
  # [from, number, module, args]
  [[-1, 1, Focus, [64, 3]],  # 1-P1/2
   [-1, 1, Conv, [128, 3, 2]],  # 2-P2/4
   [-1, 3, Bottleneck, [128]],
   [-1, 1, Conv, [256, 3, 2]],  # 4-P3/8
   [-1, 9, BottleneckCSP, [256]],
   [-1, 1, Conv, [512, 3, 2]],  # 6-P4/16
   [-1, 9, BottleneckCSP, [512]],
   [-1, 1, Conv, [1024, 3, 2]], # 8-P5/32
   [-1, 1, SPP, [1024, [5, 9, 13]]],
   [-1, 6, BottleneckCSP, [1024]],  # 10
  ]
```

Yolov5 head

```
# yolov5 head
head:
  [[-1, 3, BottleneckCSP, [1024, False]],  # 11
   [-1, 1, nn.Conv2d, [na * (nc + 5), 1, 1, 0]],  # 12 (P5/32-large)

   [-2, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 6], 1, Concat, [1]],  # cat backbone P4
   [-1, 1, Conv, [512, 1, 1]],
   [-1, 3, BottleneckCSP, [512, False]],
   [-1, 1, nn.Conv2d, [na * (nc + 5), 1, 1, 0]],  # 17 (P4/16-medium)

   [-2, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 4], 1, Concat, [1]],  # cat backbone P3
   [-1, 1, Conv, [256, 1, 1]],
   [-1, 3, BottleneckCSP, [256, False]],
   [-1, 1, nn.Conv2d, [na * (nc + 5), 1, 1, 0]],  # 22 (P3/8-small)

   [[], 1, Detect, [nc, anchors]],  # Detect(P3, P4, P5)
  ]
```

Define YOLOv5 Model Configuration and Architecture

- **custom_yolov5s.yaml**:
  - nc: **3**
  - depth_multiple: **0.33**
  - width_multiple: **0.50**

  - anchors:
    - - [10,13, 16,30, 33,23]
    - - [30,61, 62,45, 59,119]
    - - [116,90, 156,198, 373,326]
  - backbone:
    - [[-1, 1, Focus, [64, 3]], [-1, 1, Conv, [128, 3, 2]],
    - [-1, 3, Bottleneck, [128]], [-1, 1, Conv, [256, 3, 2]],
    - [-1, 9, BottleneckCSP, [256]], [-1, 1, Conv, [512, 3, 2]],
    - [-1, 9, BottleneckCSP, [512]], [-1, 1, Conv, [1024, 3, 2]],
    - [-1, 1, SPP, [1024, [5, 9, 13]]], [-1, 6, BottleneckCSP, [1024]],]

Define YOLOv5 Model Configuration and Architecture

- **head:**
  - [[-1, 3, BottleneckCSP, [1024, **False**]],
  - [-1, 1, nn.Conv2d, [na * (nc + 5), 1, 1, 0]],
  - [-2, 1, nn.Upsample, [**None**, 2, "nearest"]],
  - [[-1, 6], 1, Concat, [1]],
  - [-1, 1, Conv, [512, 1, 1]],
  - [-1, 3, BottleneckCSP, [512, **False**]],
  - [-1, 1, nn.Conv2d, [na * (nc + 5), 1, 1, 0]],
  - [-2, 1, nn.Upsample, [**None**, 2, "nearest"]],
  - [[-1, 4], 1, Concat, [1]],
  - [-1, 1, Conv, [256, 1, 1]],
  - [-1, 3, BottleneckCSP, [256, **False**]],
  - [-1, 1, nn.Conv2d, [na * (nc + 5), 1, 1, 0]],
  - [[], 1, Detect, [nc, anchors]],]

Training

- Move to the directory and use the following command to start training.
  - **$ python3 train.py --img 640 --batch 8 --epochs 30 --data ./data/score.yaml --cfg ./models/yolov5s.yaml --weights ./weights/yolov5x.pt --device 0,1**
- **Parameters**
  - — img: size of the input image
  - — batch: batch size
  - — epochs: number of epochs
  - — data: YAML file which was created in step 3
  - — cfg: model selection YAML file. I have chosen “s” for this tutorial.
  - — weights: weights file to apply transfer learning, you can find them here. https://drive.google.com/drive/folders/1Drs_Aiu7xx6S-ix95f9kNsA6ueKRpN2J
  - — device: to select the training device, **“0” for GPU**, and **“cpu”** for CPU.

Inference

- $ **python3 detect.py --source file.jpg # image**
  file.mp4 # video
  ./dir # directory
  0 # webcam

rtsp://170.93.143.139/rtplive/470011e600ef003a004ee33696235daa # rtsp stream

http://112.50.243.8/PLTV/88888888/224/3221225900/1.m3u8 # http stream

- $ **python3 detect.py --source /home/myuser/xujing/EfficientDet-Pytorch/dataset/test/ --weights weights/best.pt --conf 0.1**
- **$ python3 detect.py --source ./inference/images/ --weights weights/yolov5x.pt --conf 0.5**

- # inference 視頻
  - **$ python3 detect.py --source test.mp4 --weights weights/yolov5x.pt --conf 0.4**

訓練後的weights

- 存在Yolov5/runs/exp?/weights
- 中間檔案與數據說明

AIGO 資策會 影片

- Yolo Docker v4
  - https://www.youtube.com/watch?time_continue=2&v=HNnE65DBBoo&feature=emb_logo
- Yolo v4
  - https://www.youtube.com/watch?time_continue=4&v=tjxWp7K96Hw&feature=emb_logo
- Yolo v5
  - https://www.youtube.com/watch?v=VzKMWQKev7Q&feature=emb_logo

Demo

- How to Train YOLO v5 on a Custom Dataset
  - https://www.youtube.com/watch?v=MdF6x6ZmLAY
- Testing YOLOv5 on GTA V
  - https://www.youtube.com/watch?v=ppUFDrYc45U
- Create YOLO (v5) Dataset for Custom Object Detection using OpenCV, PyTorch and Python Tutorial
  - https://www.youtube.com/watch?v=NsxDrEJTgRw
- run yolov5 on jetson NX with own dataset
  - https://www.youtube.com/watch?v=sMTF-dngmbQ
- YOLO v5
  - https://www.youtube.com/watch?v=ptDTHla2U3o
- Spongebob prediction with yolov5
  - https://www.youtube.com/watch?v=VdjfWacnT0I

- 爭議回復
  - https://blog.roboflow.ai/yolov4-versus-yolov5/
- 如何訓練YOLOV5 (詳細) 2020/07/10
  - https://blog.roboflow.ai/how-to-train-yolov5-on-a-custom-dataset/
  - public blood cell detection dataset BCCD Dataset on Google
  - 執行範例
    - https://colab.research.google.com/github/ultralytics/yolov5/blob/master/tutorial.ipynb

- YOLOV5 介紹
  - https://blog.roboflow.ai/yolov5-is-here/
- 訓練影片？
  - https://www.bilibili.com/s/video/BV1sv411B7jK
- Demo on jetson NX
  - https://www.youtube.com/watch?v=sMTF-dngmbQ

YOLOv4 on Google Colab

Ref. https://jason-chen-1992.weebly.com/home/-google-colab-yolov4

YOLOv4 on Google Colab

- Ref. https://jason-chen-1992.weebly.com/home/-google-colab-yolov4
- **在個人的 Google Drive 上新增一個資料夾 (非必要)**
- **進入該資料夾後新增一個 Colab 文件**
- **開啟 Google Colab 然後啟用免費的GPU**

1. 首先我們先 Copy 一份 darknet 的 reo 到你的VM：
2. 下載完成後，我們需要修改 Makefile 裡面的四個參數，分別是：
   - GPU=0 要改成 GPU=1
   - OPENCV=0 要改成 OPENCV=1
   - CUDNN=0 要改成 CUDNN=1
   - CUDNN_HALF=0 要改成 CUDNN_HALF=1

```
[ ]  %cd darknet
     !sed -i 's/GPU=0/GPU=1/' Makefile
     !sed -i 's/OPENCV=0/OPENCV=1/' Makefile
     !sed -i 's/CUDNN=0/CUDNN=1/' Makefile
     !sed -i 's/CUDNN_HALF=0/CUDNN_HALF=1/' Makefile
```

```
/content/darknet
```

3. make

Step 4: Download pretrained YOLOv3 and YOLOv4 weights

YOLOv3 and YOLOv4 has been trained already on the coco dataset which has 80 classes that it can predict. We will grab these pretrained weights so that we can run YOLOv3 and YOLOv4 on these pretrained classes and get detections.

```
[ ]  download_from_official = False

     if download_from_official:
       # download weights form official
       !wget https://pjreddie.com/media/files/yolov3.weights
       !wget https://pjreddie.com/media/files/darknet53.conv.74
       !wget https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.weights
       !wget https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.conv.137
     else:
       # download weights form Jason Chen's google drive (should be faster)
       !gdown https://drive.google.com/uc?id=1hSaT4Yc19atZZulW3Q3BUDFIohfGteXN
       !gdown https://drive.google.com/uc?id=1XpMMC_eUfHKaIpfmxZa71IyocMW6ssCb
       !gdown https://drive.google.com/uc?id=1vOlvou7Pgv36l-ahej5IIfYdb_9nmy0A
       !gdown https://drive.google.com/uc?id=1Zl3rh0ZOPVj4DPZdae8WqqZU0NLX7Ncp
```

```
Downloading...
From: https://drive.google.com/uc?id=1hSaT4Yc19atZZulW3Q3BUDFIohfGteXN
To: /content/darknet/yolov3.weights
248MB [00:03, 62.3MB/s]
Downloading...
From: https://drive.google.com/uc?id=1XpMMC_eUfHKaIpfmxZa71IyocMW6ssCb
To: /content/darknet/darknet53.conv.74
162MB [00:01, 100MB/s]
Downloading...
From: https://drive.google.com/uc?id=1vOlvou7Pgv36l-ahej5IIfYdb_9nmy0A
To: /content/darknet/yolov4.weights
258MB [00:02, 101MB/s]
Downloading...
From: https://drive.google.com/uc?id=1Zl3rh0ZOPVj4DPZdae8WqqZU0NLX7Ncp
To: /content/darknet/yolov4.conv.137
170MB [00:00, 205MB/s]
```

在 weights 下載好之後，我們就可以使用：

!./darknet detect <path of cfg file> <path of weights file> <path of picture>

這個指令進行預測，預測的結果會以 "predictions.jpg" 存在跟 darknet 同一個目錄底下，為了可以直接 show 在 notebook 上面，我們可以多寫一個 imShow 的 function 來實現。

Step 5 : Run Object Detection with Darknet and YOLOv3/v4

Define the show image function

```
[ ]  def imShow(path):
       import cv2
       import matplotlib.pyplot as plt
       %matplotlib inline

       image = cv2.imread(path)
       height, width = image.shape[:2]
       resized_image = cv2.resize(image,(3*width, 3*height), interpolation = cv2.INTER_CUBIC)

       fig = plt.gcf()
       fig.set_size_inches(18, 10)
       plt.axis("off")
       plt.imshow(cv2.cvtColor(resized_image, cv2.COLOR_BGR2RGB))
       plt.show()
```

>>> 5-1. Run detection with YOLOv3

```
[ ]  # run darknet detection
     !./darknet detect cfg/yolov3.cfg yolov3.weights data/person.jpg

     # show result
     imShow('predictions.jpg')
```

>>> 5-2. Run detection with YOLOv4

```
[ ]  # run darknet detection
     !./darknet detect cfg/yolov4.cfg yolov4.weights data/dog.jpg

     # show result
     imShow('predictions.jpg')
```

Step 6 : Copy the completed darknet to Google Drive

>>> 6-1. Mount to Google Drive

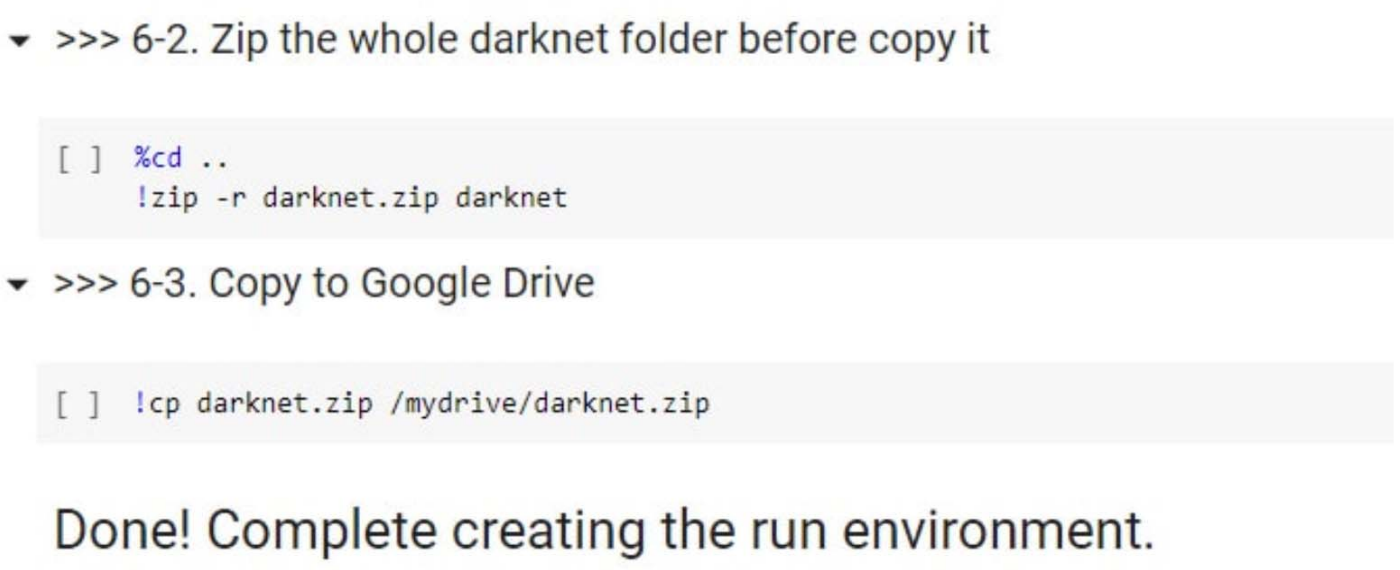
```
[ ]  from google.colab import drive
     drive.mount('/content/gdrive',force_remount=True)

     !ln -s /content/gdrive/My\ Drive/space_for_YOLO/ /mydrive
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk

Enter your authorization code:
··········
Mounted at /content/gdrive
```

在把 Google Drive 與 Google Colab VM mount 在一起後，我們就可以準備將 darknet 備份到你的 google drive 了，為了節省傳輸的時間我們可以先把整個 darknet 資料夾壓縮起來，在壓縮完成後就可用 cp 指令把它複製進 goole drive 搂~



Step 8 開始訓練訓練指令

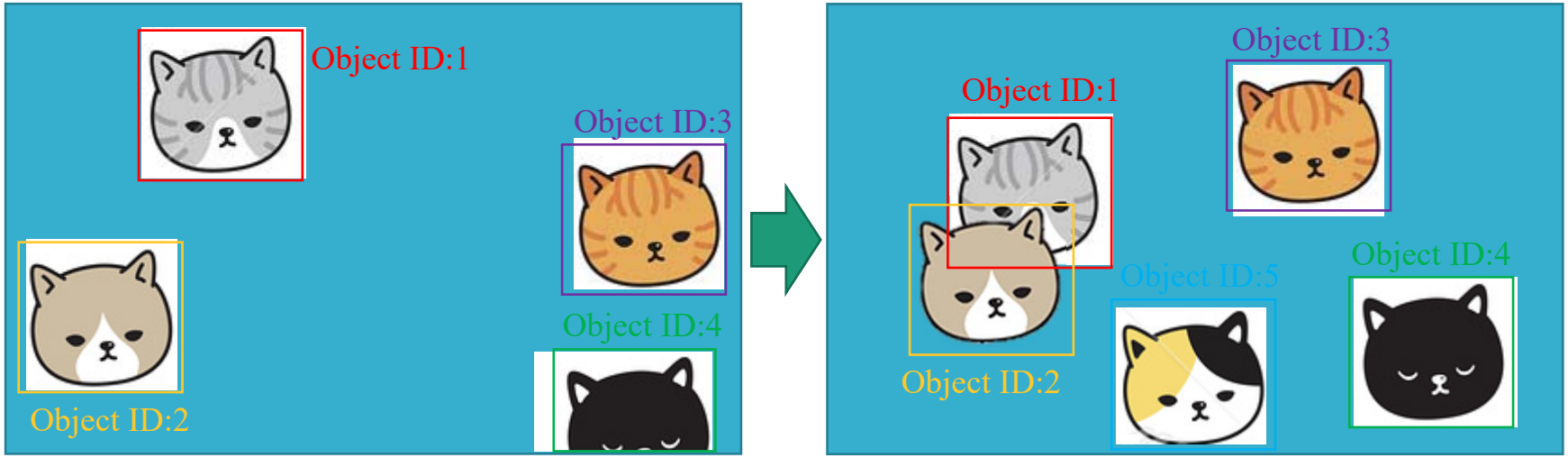
- $ ./darknet detector train data/obj.data yolo-obj.cfg yolov4.conv.137
- 在**data**裡面提供的就是圖片的路徑，每一行就是一張圖片，
  - **cfg**裡面就是網路結構的定義。
  - 准備的就是這兩個檔了，接著就可以開始訓練
- 請注意，如果你不是resume訓練，那麼可以從yolov4官方repo裡面下載對應的conv73的weights進行開始訓練。

https://zhuanlan.zhihu.com/p/138791419?fbclid=IwAR3WBZjfGEdCn_Tx87WlGwM5REeyg8DoZ1ZxohvciYdIr4N_p9wFFHlssrU

What is Tracker?

What is Tracker?

- Justify if the objects of the adjacent frame belong to the same one or not.
- Same object detected ➡ Assign the same object ID
- New object detected ➡ Assign a new object ID

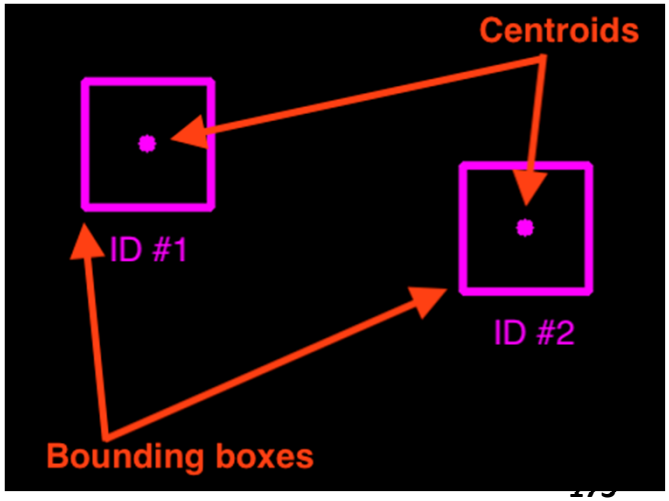


frame n

frame n+1

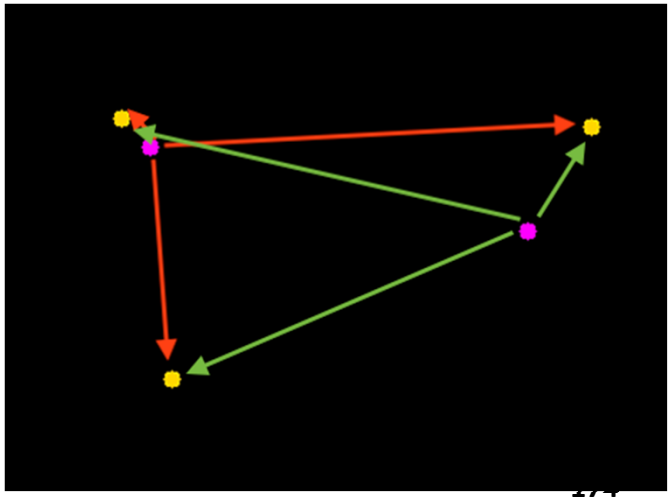
Centroid Tracking

- **Step 1**: Accept the coordinates of bounding box and compute **centroids**
  - A set of bounding box (*x*, *y*)-coordinates are read for each detected object in each frame.
  - The centroid (the center (*x*, *y*)-coordinates) of the bounding box coordinates will be computed.
  - The first initial set of bounding boxes will have unique IDs.



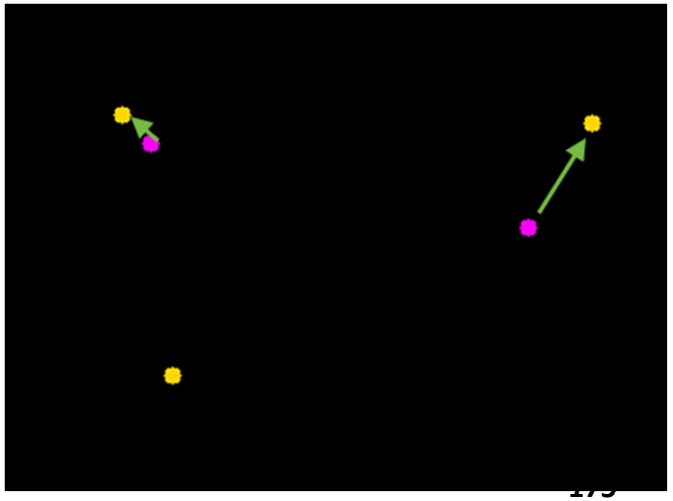
Centroid Tracking

- **Step 2**: Compute **Euclidean distance** between new bounding boxes and existing objects
  - To associate the **new object centroids** with the **old object centroids**, we need to compute the **Euclidean distance** between each pair of the existing object centroids and the input object centroids.



Centroid Tracking

- **Step 3**: Update (*x*, *y*)-coordinates of the existing objects
  - The primary assumption of the centroid tracking algorithm is that a given object will **potentially move in between subsequent frames**.
  - Associate centroids with **minimum distances** between subsequent frames are chosen to build an object tracker.

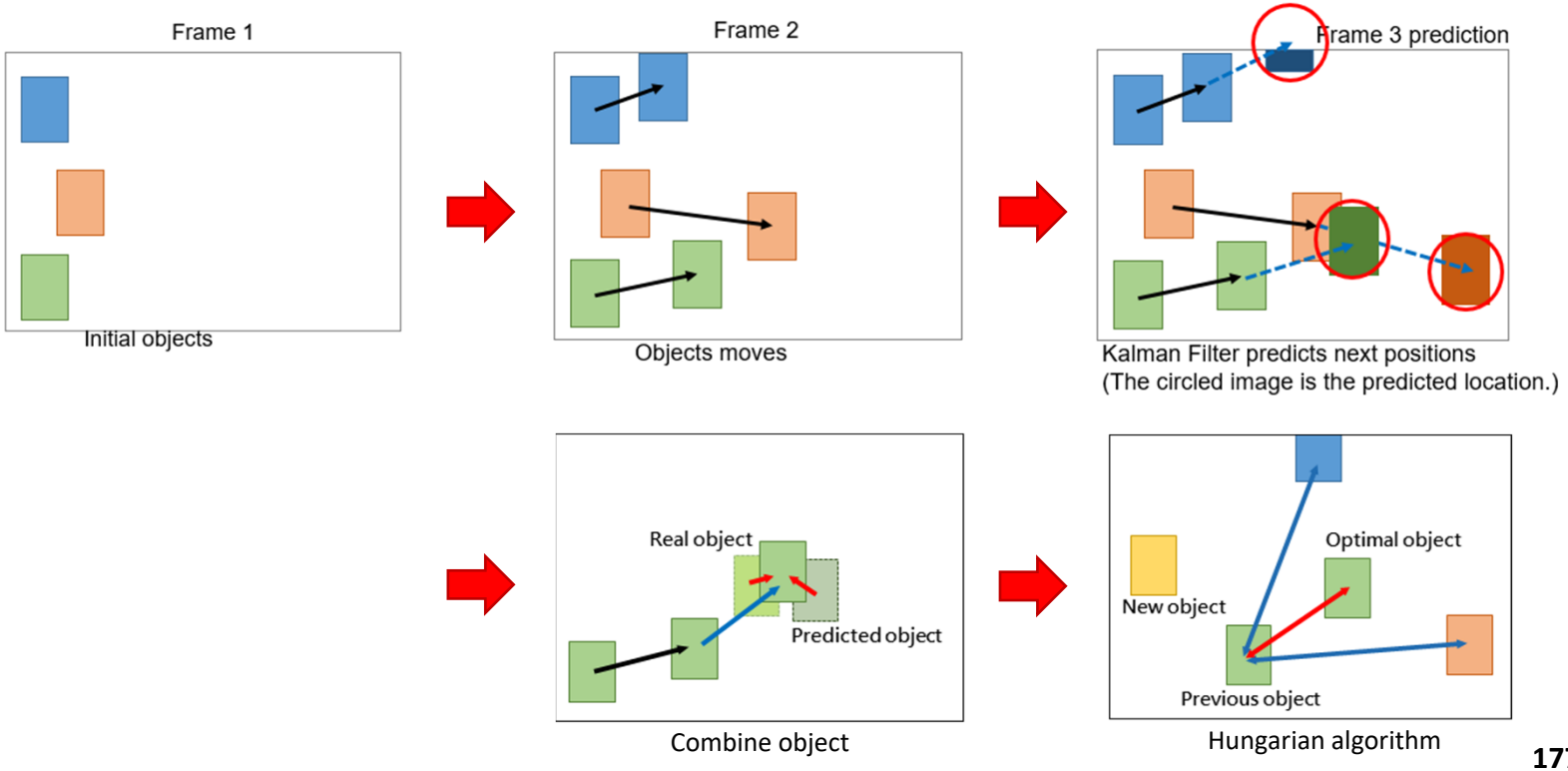


Centroid Tracking

- **Step 4**: Register new objects
  - If more input objects are detected, the new objects are registered and assigned a new ID.
  - Then the algorithm goes back to **Step 2** and repeats.

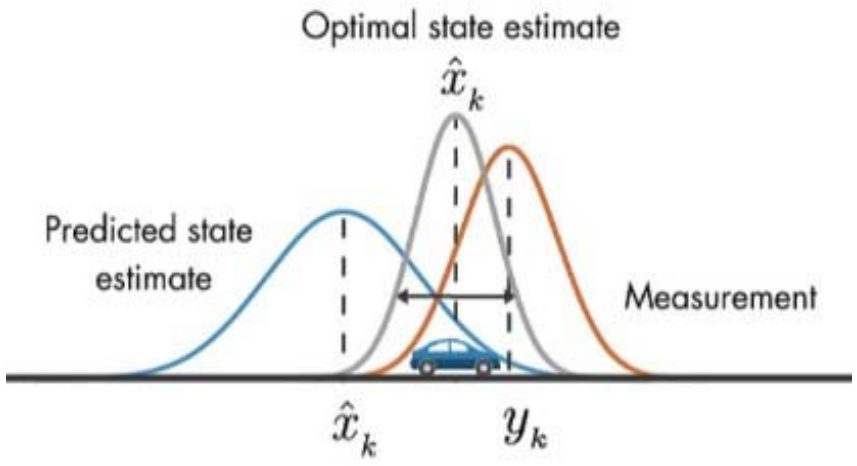
SORT

- SORT (Simple Online and Realtime Tracking)
  - Kalman Filter
  - Hungarian Algorithm

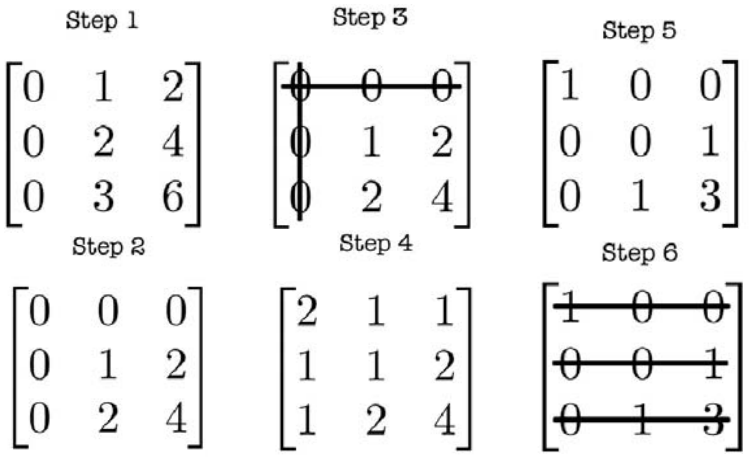


SORT

- **Kalman Filter**:
  - Predict the future location of the objects.
- **Hungarian Algorithm**:
  - Match the same objects with adjacent frames.
  - 在多項式時間內求解任務分配問題的組合最佳化演算法，



Kalman Filter



Hungarian Algorithm

演算法概念

- 要理解SORT算法，首先需要明確論文中提到的兩個名詞：Tracks和Detections。
  - Tracks: 是指在已經匹配成功的所有目標狀態量（估計值），正是通過Tracks才能進行卡爾曼濾波的預測。
  - Detections: 是通過目標檢測器獲取的當前幀的檢測框（觀測值）

演算法概念

- 對於目標狀態量，在視頻中，每個移動的目標可以用一個目標框進行表示，每個目標框可以由以下變量進行表示：
- X= [u, v, s, r, u’, v’, s’]

- 其中
  - u, v 表示目標在圖中的中心位置坐標，
  - s 表示目標框的面積大小，
  - r 表示目標框的長寬比。
  - u’, v’ s’ 分別表示位置、面積的變化速率。

- 設當前我們已經得到了某一目i在1:t-1時刻的所有狀態的估計值，記為 $\hat{x}^i_{1:t-1}$ ，0≤I≤m
- 我們也得到了當前時刻 t 的目標檢測器檢測到的 n 個目標的觀測量，記為 $\tilde{z}^j_t$ ，0≤j≤n

- 目標檢測器無法直接得到目標的速率變化量，因此觀測量的向量表示為：
  - z=[u, v, s, r]

- 從x->z 的轉換可用矩陣 H 完成：

$$\hat{z}_k^i = H\hat{x}_k^i$$

- 接下來說明如何利用卡爾曼濾波與匈牙利算法完成 狀態的關聯。

$$\hat{x}_t^j \rightarrow x_{1:t-1}^i$$

卡爾曼濾波器演算法(補充)

- 首先介紹一個離散控制過程的系統，該系統可以用一個線性隨機微分方程式(Linear Stochastic equation)來表示：

$$X(k) = F \times X(k-1) + B \times U(k) + W(k)$$

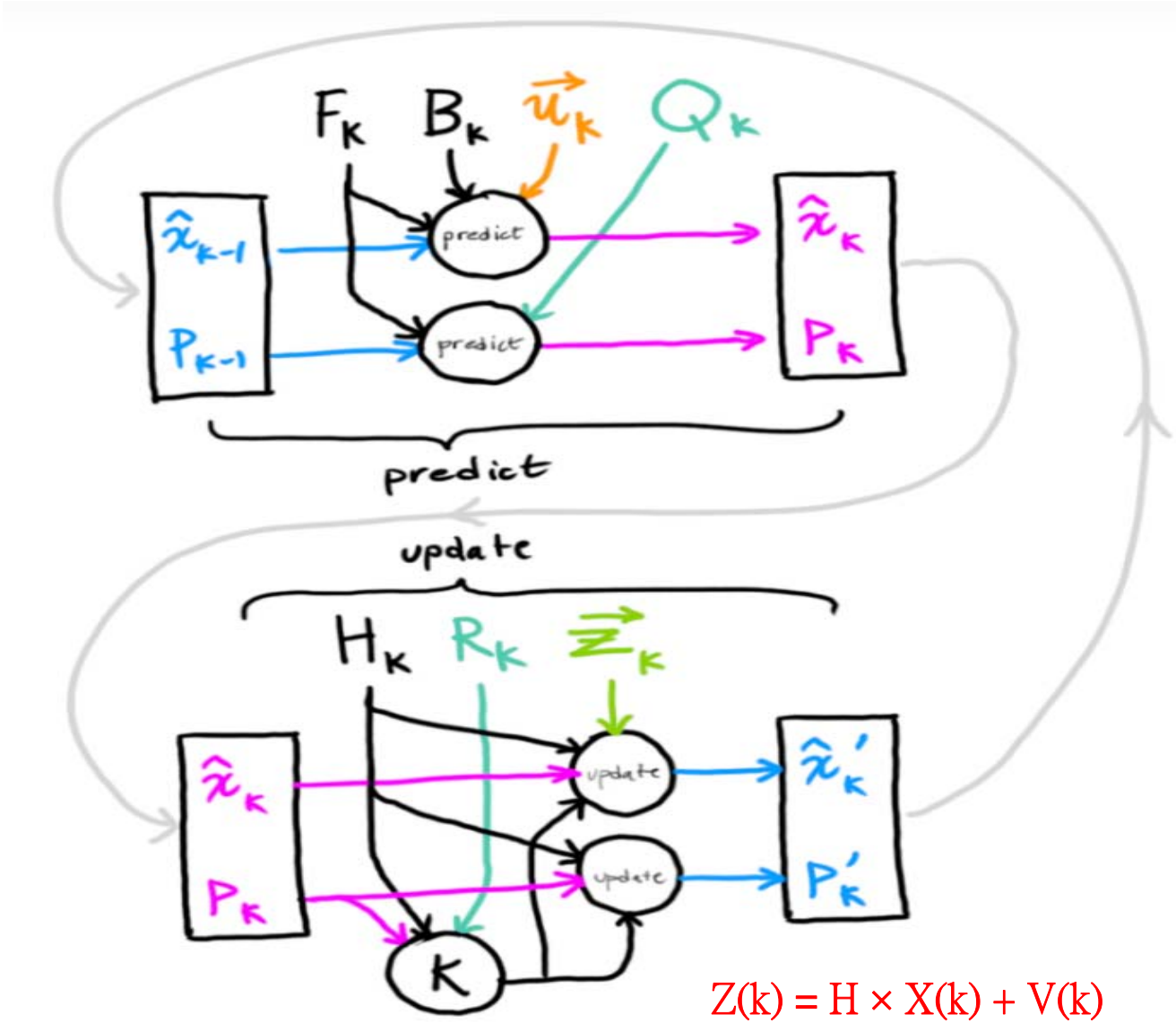
該系統的測量值為：

$$Z(k) = H \times X(k) + V(k)$$

其中：

- X(k) 是 k 時刻的**系統狀態**
- U(k) 是 k 時刻對系統的**控制量**
- F 和 B 是系統參數，對於多模型系統，他們是矩陣
- Z(k) 是 k 時刻的 **測量值**
- H 是測量系統的參數，對於多測量系統，H 是矩陣
- W(k) 與 V(k) 分別表示**過程和測量的雜訊**，被假設為高斯白雜訊，covariance **分是** Q **及** R，這裡假設他們不會隨系統狀態變化而變化

$$X(k) = F \times X(k-1) + B \times U(k) + W(k)$$



$$Z(k) = H \times X(k) + V(k)$$

https://zhuanlan.zhihu.com/p/378388675

首先，利用式(3)将 $\mathbf{Tracks}_{t-1}$ 中的每个目标状态转换为观测量的维度（左乘矩阵H相当于截取x的前四维变量），得到预测值状态量$\hat{z}_i^t, 1 \leq i \leq m$。

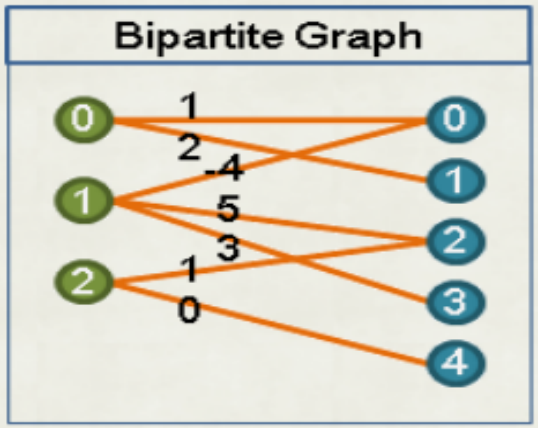
假设在时刻 t 的目标的真实状态是 z^t， 现在我们有在时刻 t 的关于目标的估计量 $\mathbf{Tracks}_t = \{\hat{z}_i^t, 1 \leq i \leq m\}$，也有在时刻 t 的观测量 $\mathbf{Detections}_t = \{\tilde{z}_j^t, 1 \leq j \leq n\}$。利用IOU来评价Tracks与Detections两两目标检测的距离，从而得到 IOU 关联矩阵。例如，当前时刻得到的IOU关联矩阵（原文中设置了一个IOU阈值来过滤）如下所示：

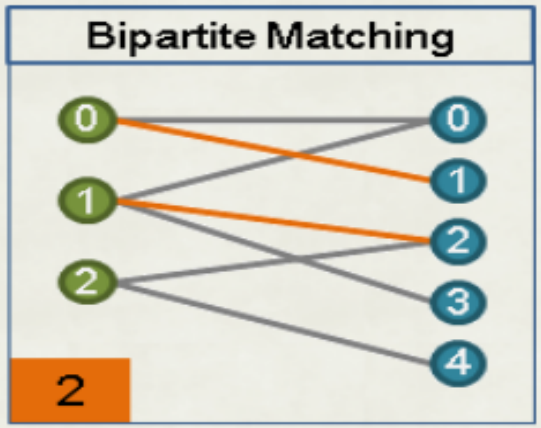
$$iou = \begin{bmatrix} 0.85 & 0 & 0 & 0.08 & 0 \\ 0 & 0.79 & 0 & 0 & 0 \\ 0.21 & 0 & 0 & 0.64 & 0 \\ 0 & 0 & 0 & 0 & 0.94 \\ 0 & 0 & 0.50 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

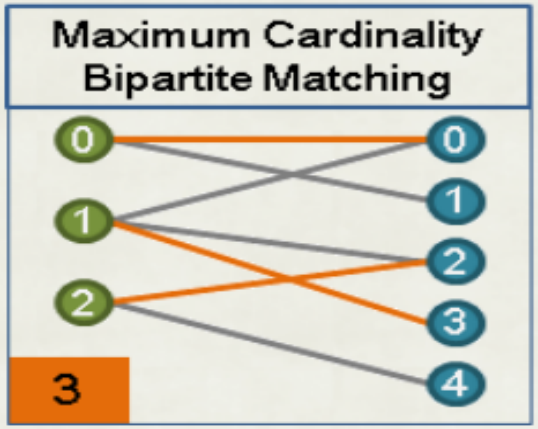
其中行表示$\mathbf{Detections}$，列表示$\mathbf{Tracks}$，我们的目标是将每个Detection分配到已知的Tracks中，采用的目标函数是使得到的相似度之和最大，约束条件是同一个Detection不能同时分给两个Tracks，同一个Tracks也不能接收两个Detection。这种分配问题可以交给匈牙利算法来解决。

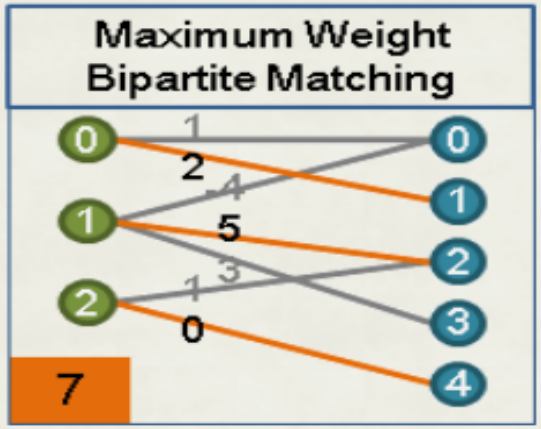
Bipartite Matching

一張二分圖上的匹配，稱作「二分匹配」。所有的匹配邊，都是這橫跨這兩群點的邊，就像是連連看。









https://web.ntnu.edu.tw/~algo/Matching.html

情況處理

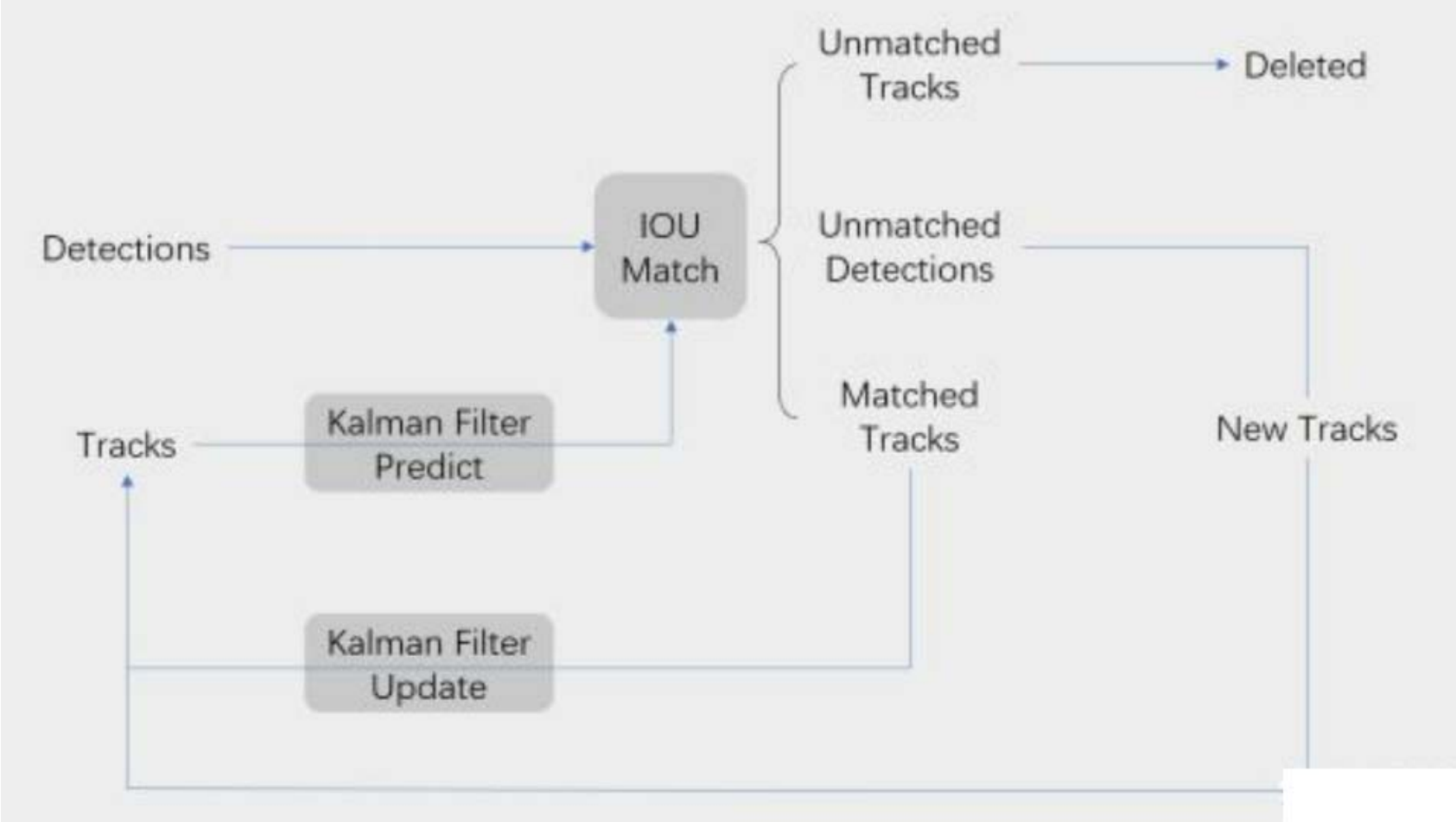
- 當Detections大於Tracks時，
  - 說明有新的目標出現，在當前幀肯定有Detection無法分配給Tracks，這種可以叫做unmatched detections。原文采用的策略是新建一個新的Track。
- 當Detections等於Tracks時
  - 最理想情況，能夠將Detection與Tracks進行一一匹配。
- 當Detections小於Tracks時，
  - 說明有目標消失或被遮擋，在當前幀肯定有Track無法接收Detections，這種track可以叫做unmatched track。
  - 原文采用的策略是設置一個 T_{lost}，若目標在連續的幀中不出現，那麼將該Track刪除。

狀態更新

- 最後一步，利用卡爾曼濾波的狀態更新方程對t時刻的x狀態進行更新:

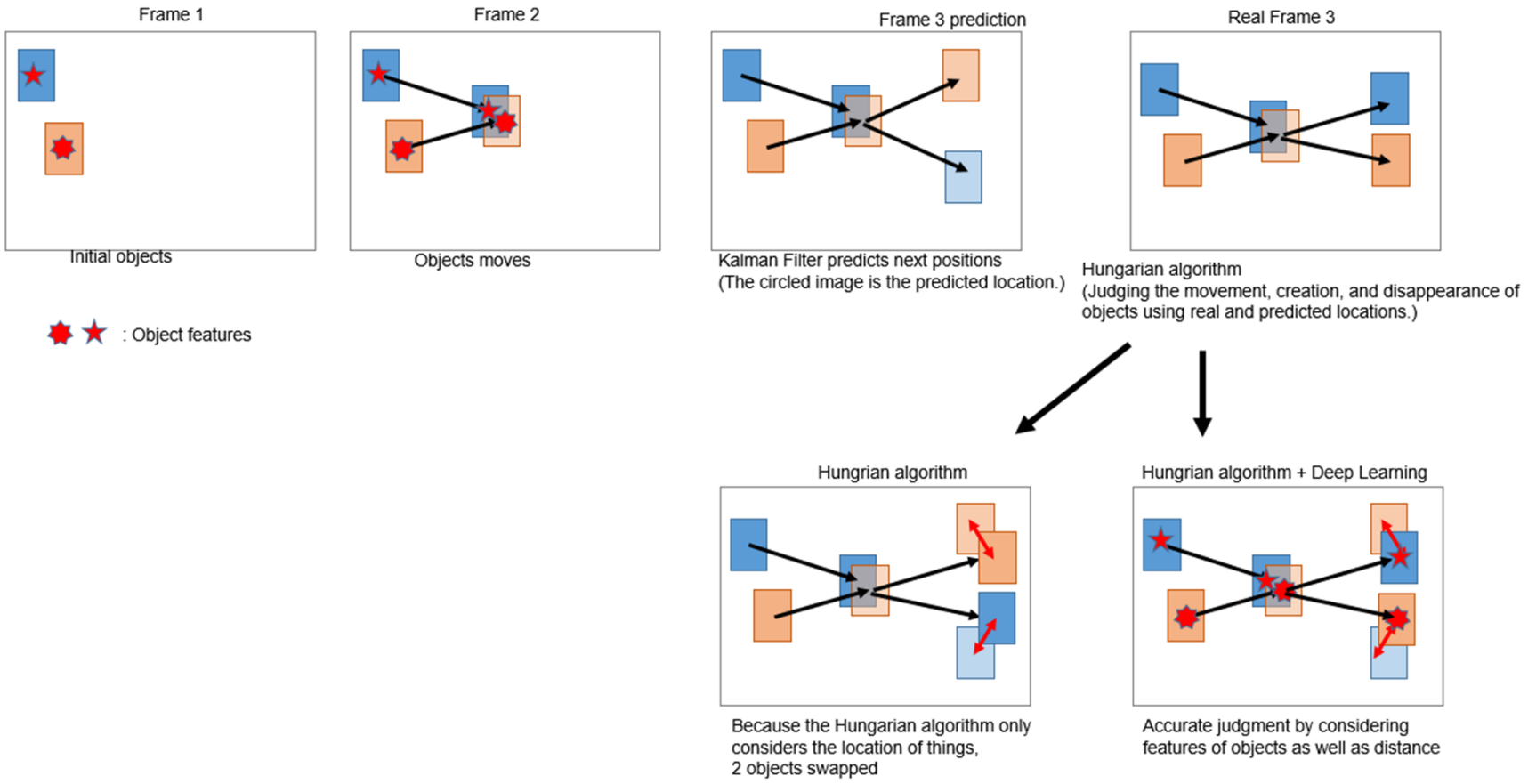
$$\hat{x}'_k = \hat{x}_k + K'(z_k - H_k\hat{x}_k)$$
$$P'_k = P_k - K'H_kP_k$$
$$where\ K' = P_kH_k^T(H_kP_kH_k^T + R_k)^{-1}$$

流程圖


Unmatched
Tracks
Deleted
Detections
IOU
Match
Unmatched
Detections
Matched
Tracks
New Tracks
Tracks
Kalman Filter
Predict
Kalman Filter
Update

Deep SORT

- Deep SORT
  - Reduce ID switch issue on SORT.



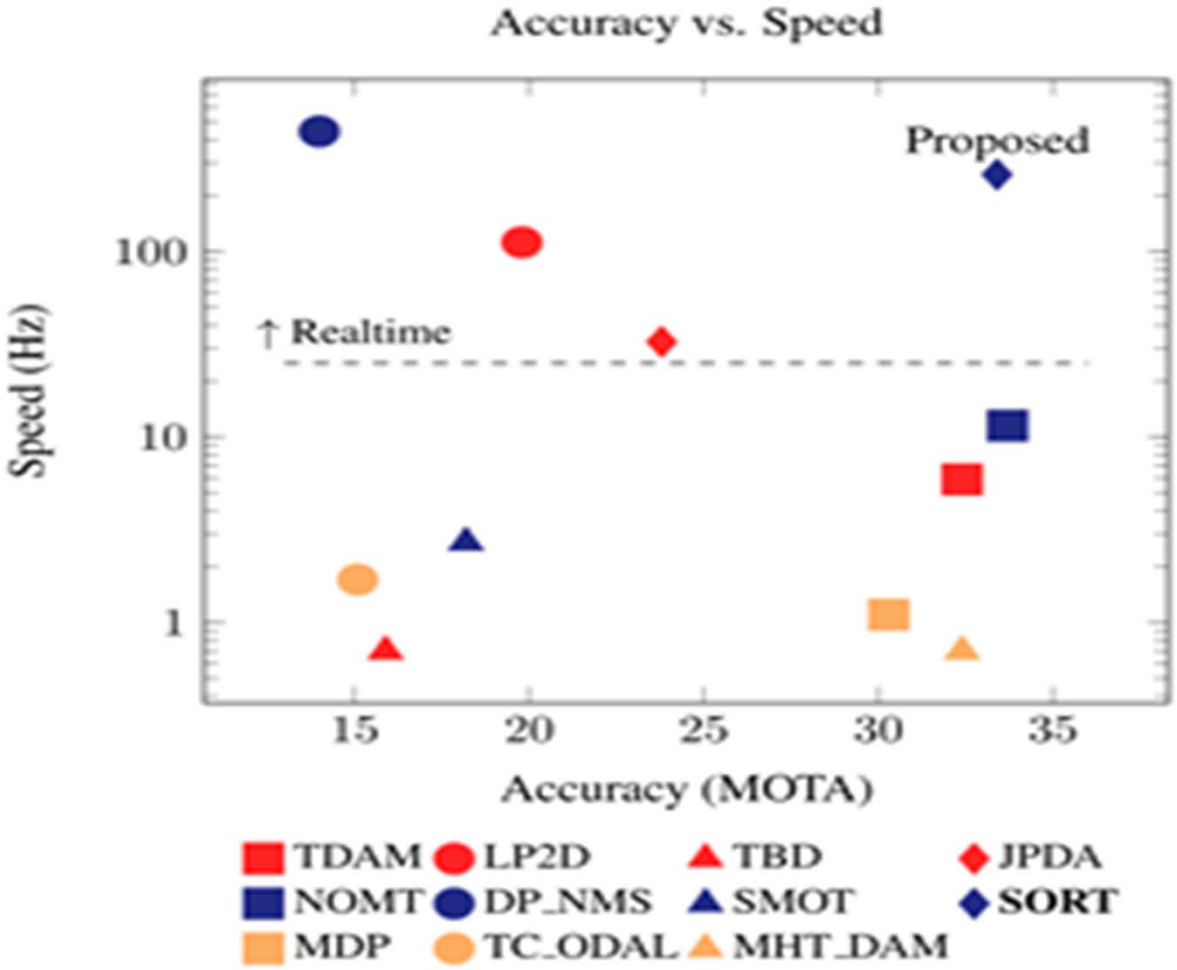


Fig. 1. Benchmark performance of the proposed method (**SORT**) in relation to several baseline trackers [6]. Each marker indicates a trackers accuracy and speed measured in frames per second (FPS) [Hz], i.e. higher and more right is better.

https://blog.csdn.net/qq_41214679/article/details/110276301

- **Reference**

- Bewley, Alex, Zongyuan Ge, Lionel Ott, Fabio Ramos和Ben Upcroft. 《[SORT]Simple Online and Realtime Tracking》. *2016 IEEE International Conference on Image Processing (ICIP)*, 2016年9月, 3464–68. https://doi.org/10.1109/ICIP.2016.7533003.
- Leal-Taixé, Laura, Anton Milan, Ian Reid, Stefan Roth和Konrad Schindler. 《MOTChallenge 2015: Towards a Benchmark for Multi-Target Tracking》. *arXiv:1504.01942 [cs]*, 2015年4月8日. http://arxiv.org/abs/1504.01942.