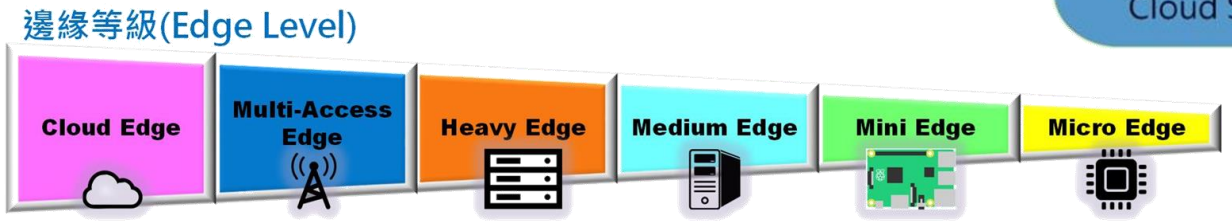
國立臺灣科技大學
NATIONAL TAIWAN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Department of Computer Science & Information Engineering
資 訊 工 程 系

# 人工智慧與邊緣運算實務

## 7.2 邊緣智慧案例實作【物件偵測】

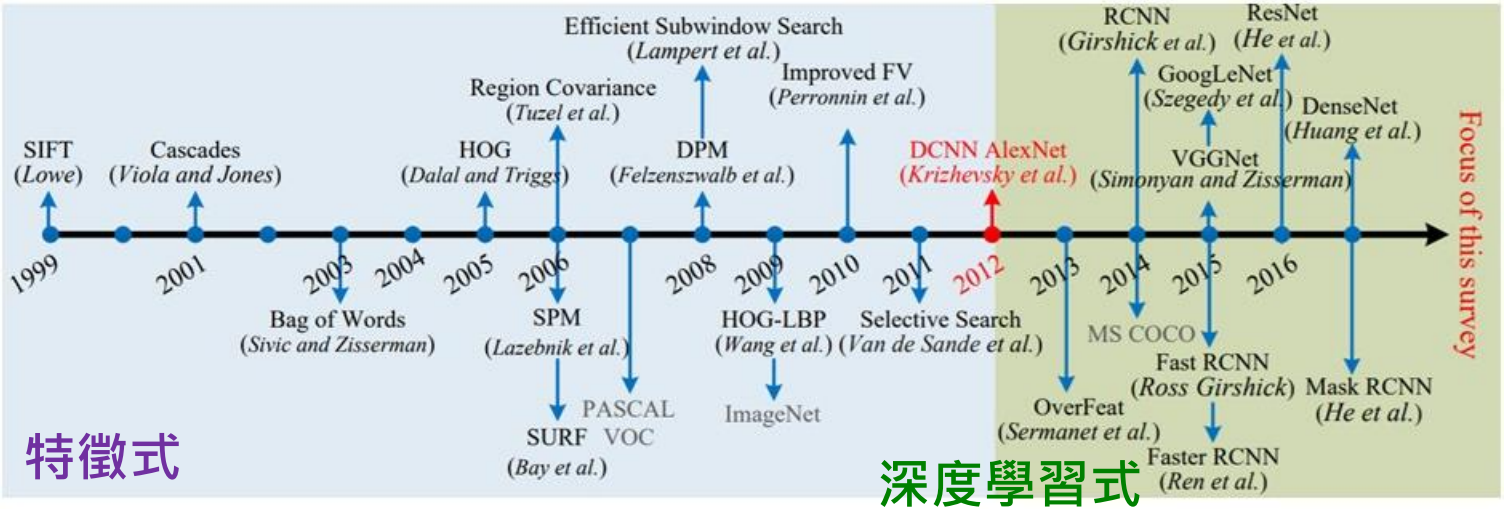



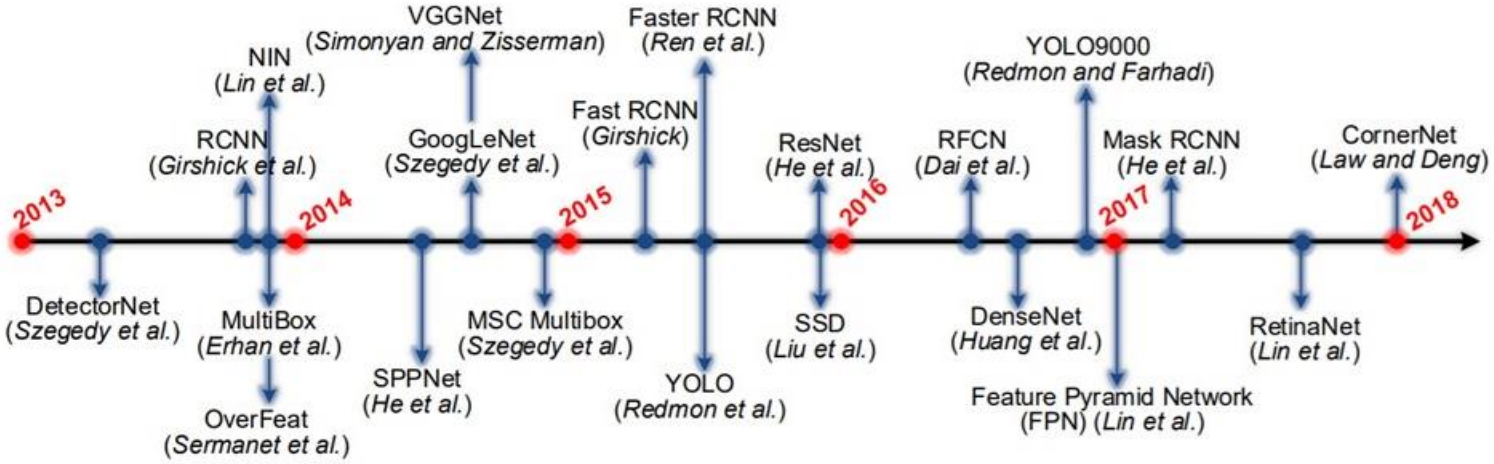

資訊工程系　許哲豪 助理教授

# 7.2 物件偵測

- 物件偵測簡介
- 預訓練模型推論
- 自定義模型訓練及推論

# 物件偵測技術演進



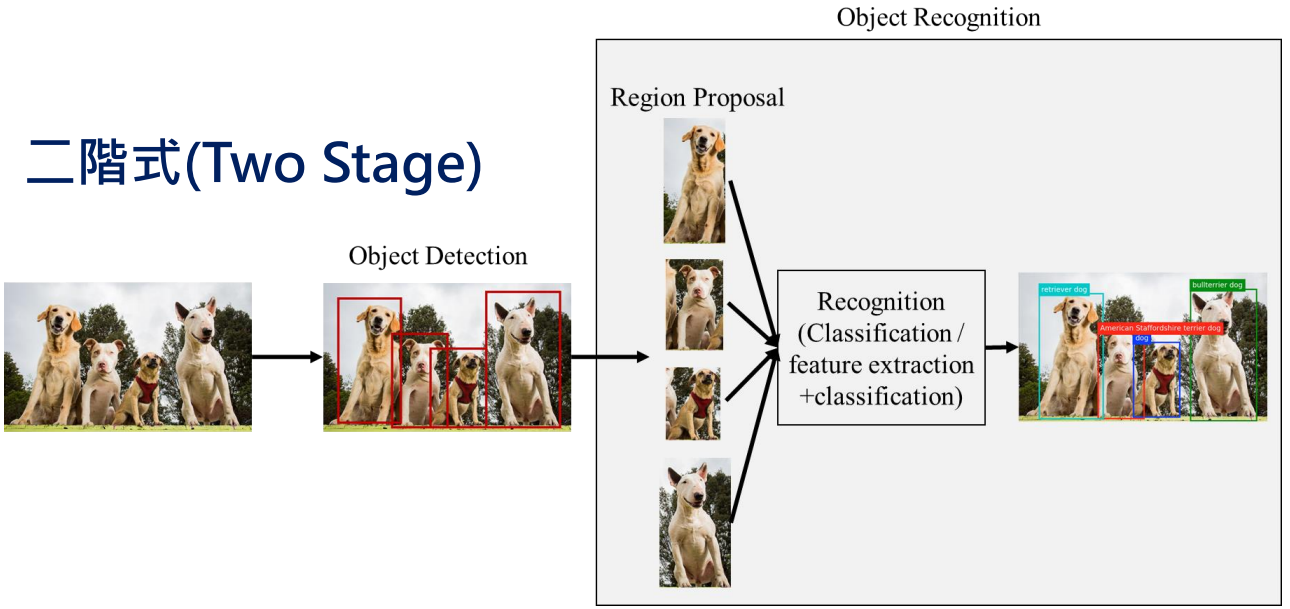

**特徵式**
精度低速度快

**二段式**
精度高速度慢

**一段式**
精度可速度快

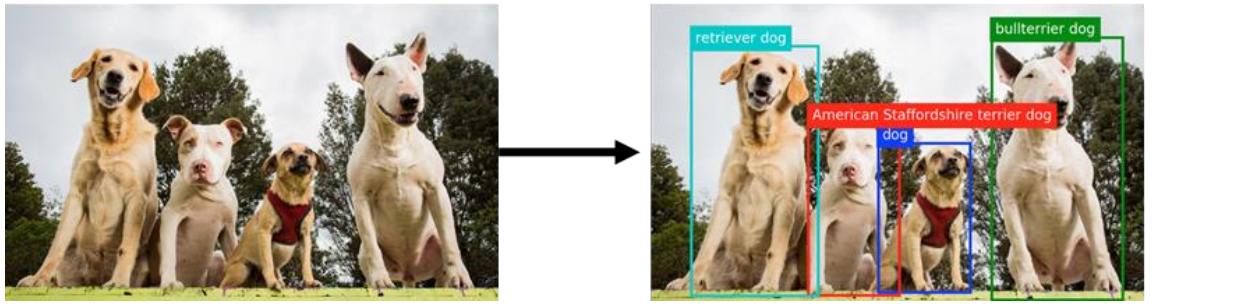
# 常見物件偵測模型

## 二階式(Two Stage)




先找出物件可能區域(Region Proposals)再進行物件辨識，如
R-CNN
Fast R-CNN
Faster R-CNN
…

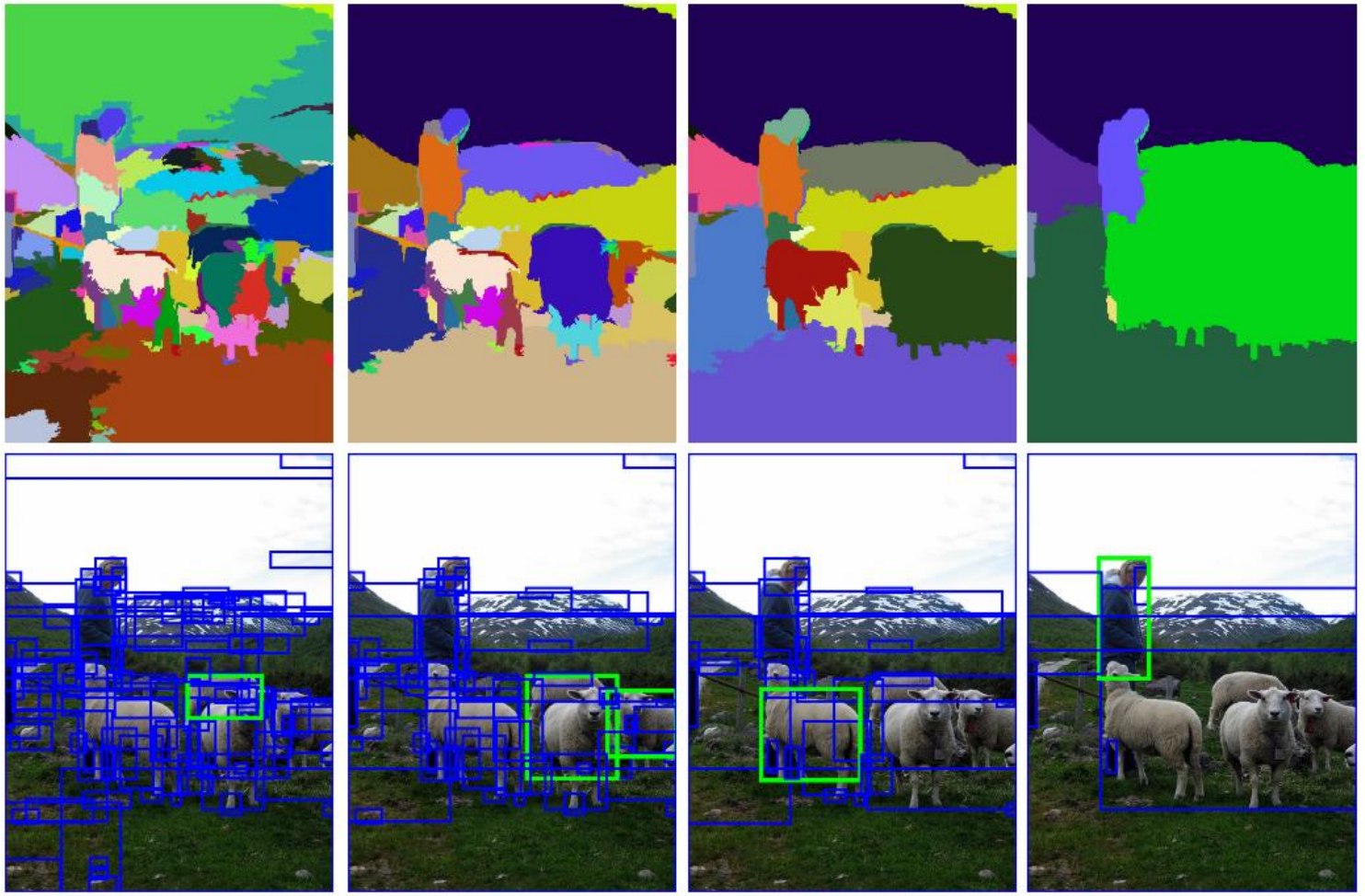
## 一階式(One Stage)




物件偵測及辨識一起完成，如
SSD
YOLO
…

# 候選區域(Region Proposals)

## Selective Search for Object Recognition

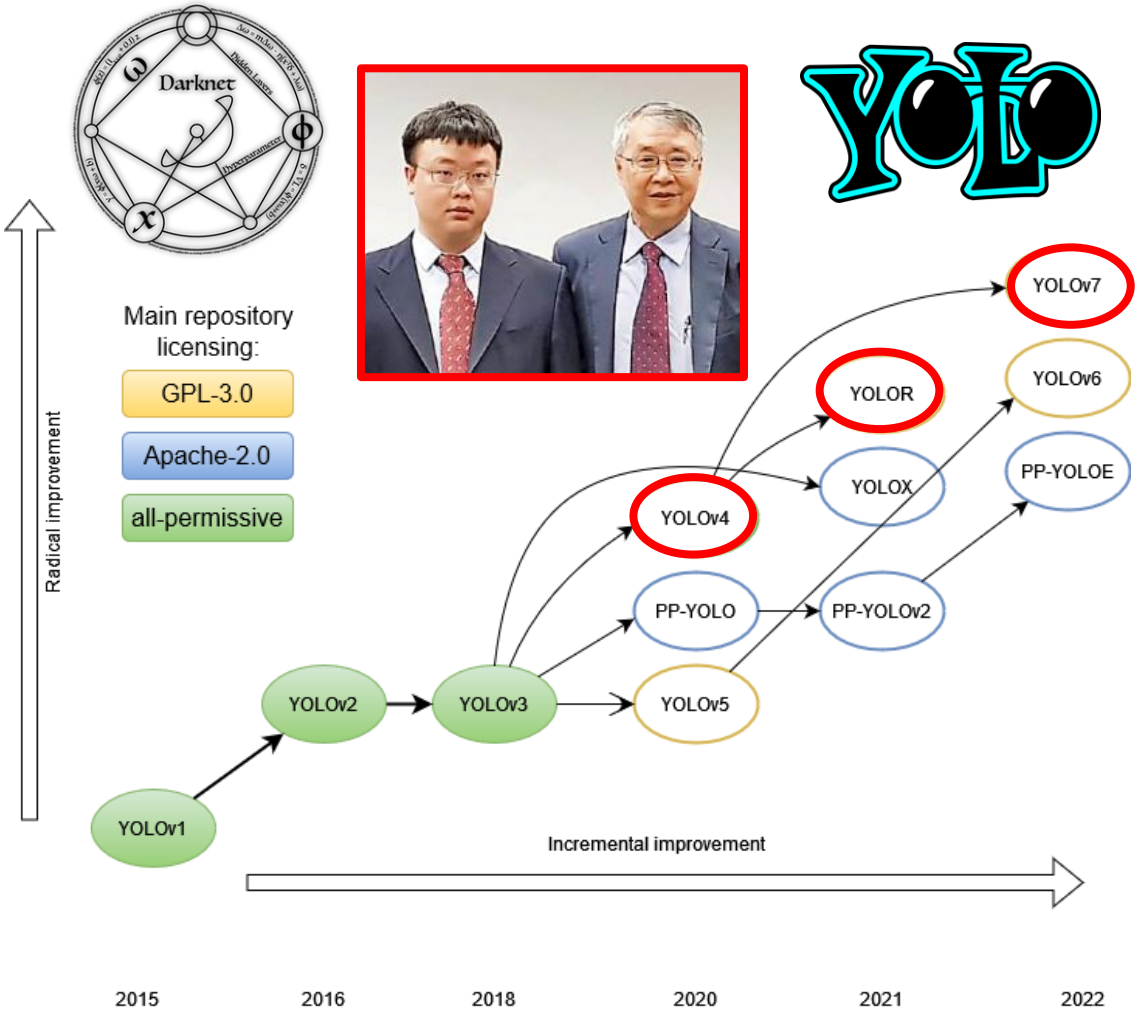


**二階式物件偵測**

**第一階：**產生候選區域再合併。

資料來源：http://www.huppelen.nl/publications/selectiveSearchDraft.pdf

# Yolo 物件偵測模型發展歷程




## 一階式物件偵測

## You Only Look Once

- 簡稱YOLO，目前已發展至八代(v8)，但其中v5, v6, v8為商用版本，非原團隊發表之學術論文。
- 其中v4及v7為中研院資訊所廖弘源所長及其學生王建堯博士和原團隊Alexey Bochkovskiy共同開發，深獲全世界肯定，大量使用在各個領域。

影像來源：https://medium.com/deelvin-machine-learning/the-evolution-of-the-yolo-neural-networks-family-from-v1-to-v7-48dd98702a3d

# OpenVINO Model Zoo (1/3)

| Model Name | Complexity (GFLOPs) | Size (Mp) |
|---|---|---|
| faster-rcnn-resnet101-coco-sparse-60-0001 | 364.21 | 52.79 |
| face-detection-adas-0001 | 2.835 | 1.053 |
| face-detection-retail-0004 | 1.067 | 0.588 |
| face-detection-retail-0005 | 0.982 | 1.021 |
| face-detection-0200 | 0.785 | 1.828 |
| face-detection-0202 | 1.767 | 1.842 |
| face-detection-0204 | 2.405 | 1.851 |
| ⋮ | ⋮ | ⋮ |
| yolo-v2-tiny-ava-sparse-60-0001 | 6.975 | 15.12 |
| yolo-v2-tiny-vehicle-detection-0001 | 5.424 | 11.229 |
| smartlab-object-detection-0001 | 1.077 | 0.8908 |
| smartlab-object-detection-0002 | 1.073 | 0.8894 |
| smartlab-object-detection-0003 | 1.077 | 0.8908 |
| smartlab-object-detection-0004 | 1.073 | 0.8894 |

60個以上
**Intel** Pre-Trained模型

**OpenVINO 2022.3**
https://docs.openvino.ai/latest/omz_models_group_intel.html#object-detection-models

# OpenVINO Model Zoo (2/3)

| Model Name | Implementation | OMZ Model Name |
|---|---|---|
| CTPN | TensorFlow* | ctpn |
| CenterNet (CTDET with DLAV0) 512x512 | ONNX* | ctdet_coco_dlav0_512 |
| DETR-ResNet50 | PyTorch* | detr-resnet50 |
| EfficientDet-D0 | TensorFlow* | efficientdet-d0-tf |
| EfficientDet-D1 | TensorFlow* | efficientdet-d1-tf |
| FaceBoxes | PyTorch* | faceboxes-pytorch |
| SSD with ResNet 34 1200x1200 | PyTorch* | ssd-resnet34-1200-onnx |
| Ultra Lightweight Face Detection RFB 320 | PyTorch* | ultra-lightweight-face-detection-rfb-320 |
| Ultra Lightweight Face Detection slim 320 | PyTorch* | ultra-lightweight-face-detection-slim-320 |
| Vehicle License Plate Detection Barrier | TensorFlow* | vehicle-license-plate-detection-barrier-0123 |

30個以上
**Public** Pre-Trained模型

**OpenVINO 2022.3**

https://docs.openvino.ai/latest/omz_models_group_public.html#object-detection-models

# OpenVINO Model Zoo (3/3)

**OpenVINO 2022.3**

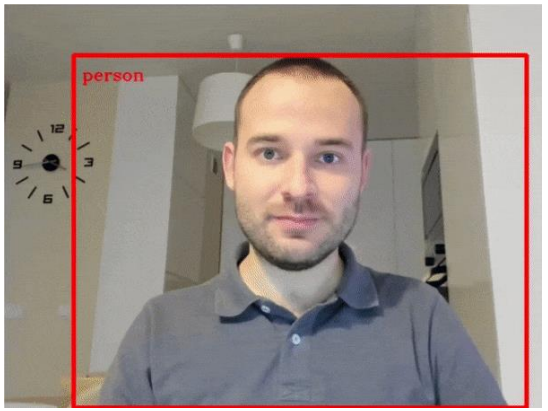
**YOLO家族相關**

| Model Name | Implementation | OMZ Model Name | Accuracy | GFlops |
|---|---|---|---|---|
| YOLO v1 Tiny | TensorFlow.js* | yolo-v1-tiny-tf | 54.79% | 6.9883 |
| YOLO v2 Tiny | Keras* | yolo-v2-tiny-tf | 27.3443%/29.1184% | 5.4236 |
| YOLO v2 | Keras* | yolo-v2-tf | 53.1453%/56.483% | 63.0301 |
| YOLO v3 | Keras* ONNX* | yolo-v3-tf yolo-v3-onnx | 62.2759%/67.7221% 48.30%/47.07% | 65.9843~65.998 |
| YOLO v3 Tiny | Keras* ONNX* | yolo-v3-tiny-tf yolo-v3-tiny-onnx | 35.9%/39.7% 17.07%/13.64% | 5.582 |
| YOLO v4 | Keras* | yolo-v4-tf | 71.23%/77.40% /50.26% | 129.5567 |
| YOLO v4 Tiny | Keras* | yolo-v4-tiny-tf | | 6.9289 |
| YOLOF | PyTorch* | yolof | 60.69%/66.23% /43.63% | 175.37942 |
| YOLOX Tiny | PyTorch* | yolox-tiny | 47.85%/52.56% /31.82% | 6.4813 |

https://docs.openvino.ai/latest/omz_models_group_public.html#object-detection-models

# Ex1: OpenVINO Notebooks 401

- Tutorials Notebooks Live Demos 401-object-detection-webcam




1. 進入命令列模式執行程式

**jupyter 401-object-detection.ipynb**

2. 本程式使用微軟COCO資料集預訓練之SSDLite MobileNetV2模型。
3. 本程式支援使用webcam或mp4影片檔

**video_file = "../201-vision-monodepth/data/Coco Walking in Berkeley.mp4"**

**run_object_detection(source=video_file, flip=False, use_popup=False)**

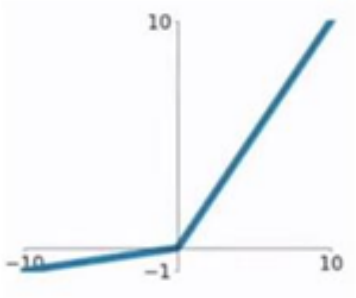
**ps. source=0 時，表示使用電腦上第一組網路攝影機。**

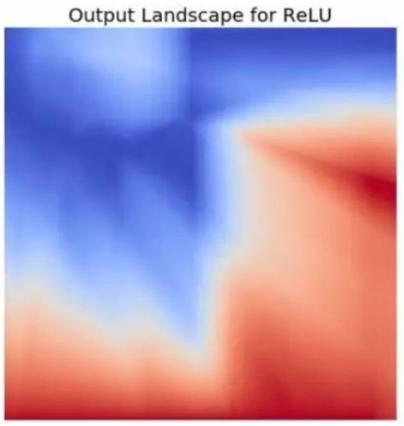
資料來源：https://docs.openvino.ai/2022.1/notebooks/401-object-detection-with-output.html

# YOLOv4 Model Zoo

| Model | Backbone | Neck | Plugin | V100 FPS | BFLOPs |
|---|---|---|---|---|---|
| **YOLOv4** | CSPDarknet53 with Mish Activation | PANet with Leaky activation | SPP | 62@608x608, 83@512x512 | 128.5@608x608 |
| **YOLOv4-Leaky** | CSPDarknet53 with Leaky activation | PANet with Leaky activation | SPP | | 128.5@608x608 |
| **YOLOv4-SAM-Leaky** | CSPDarknet53 with Leaky activation | PANet with Leaky activation | SPP, SAM | | 130.7@608x608 |
| **YOLOv4-Mish** | CSPDarknet53 with Mish activation | PANet with Mish activation | SPP | | 128.5@608x608 |
| **YOLOv4-SAM-Mish** | CSPDarknet53 with Mish activation | PANet with Mish activation | SPP, SAM | 61@608x608, 81@512x512 | 130.7@608x608 |
| **YOLOv4-CSP** | CSPDarknet53 with Mish activation | CSPPANet with Mish activation | SPP | | |
| **YOLOv4-CSP-SAM** | CSPDarknet53 with Mish activation | CSPPANet with Mish activation | SPP, SAM | | |

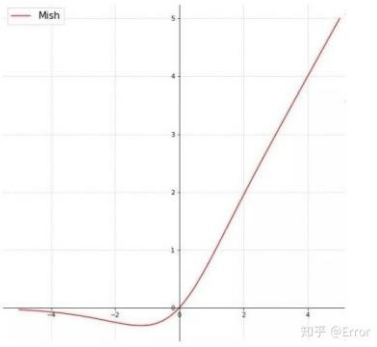
**Leaky ReLU**





**Mish=x * tanh(ln(1+e^x))**




資料來源：https://github.com/AlexeyAB/darknet/wiki/YOLOv4-model-zoo

# 預訓練模型

- Yolov4使用MS COCO(80分類)資料集
- 預訓練權重檔下載
  - yolov4x-mish.weights(381 MB)
  - yolov4-csp.weight(202 MB)
  - yolov4.weights(245 MB)
  - yolov4-tiny.weights(23.1 MB)
  - enetb0-coco_final.weights(18.3 MB)
  - yolov3-openimages.weights(247 MB)

https://github.com/AlexeyAB/Darknet#pre-trained-models

# Ex2: Yolov4-tiny + Webcam測試

- 直接從Github運行Colab範例

https://colab.research.google.com/github/OmniXRI/Yolov4_Colab_User_Datasets/blob/main/Colab_Yolov4_Webcam_Test.ipynb

- 主要步驟：

1. 驗證Nvidia GPU及CUDA版本（可略）
2. 下載DarkNet及Yolov4-tiny預訓練權重檔
3. 修改Make參數
4. 編譯Darknet
5. 測試darknet編譯結果
6. 建立Javascript輔助函式
7. 測試從網路攝影機取得靜態影像並進行物件偵測
8. 建立動態影像處理Javascript函式
9. 從網路攝影機取得動態影像並進行物件偵測

# Ex3: 自定義資料集範例程式說明

- 從Github下載範例程式（Yolov4 & Yolov4-tiny版）

https://github.com/OmniXRI/Yolov4_Colab_User_Datasets

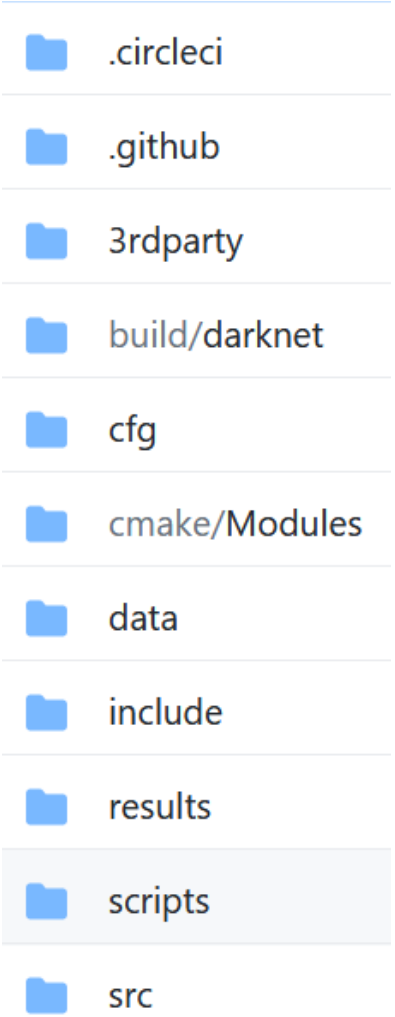
https://github.com/OmniXRI/Yolov4-tiny_Colab_User_Datasets

- yolov4(_tiny)_training_test.ipynb 為主程式，包含Yolov4(Yolov4_tiny)**預訓練測試**、**自定義資料集訓練及推論測試**。

https://colab.research.google.com/github/OmniXRI/Yolov4-tiny_Colab_User_Datasets/blob/main/yolov4_tiny_training_test.ipynb

- my_dataset.zip 包含100張影像及Yolo格式標註檔(*.txt)
- 在個人的Google雲端硬碟新增一個yolov4的路徑，將下載到的所有檔案上傳至雲端硬碟yolov4中。

# Darknet檔案夾結構




從Github下載darknet
https://github.com/AlexeyAB/darknet

**組態設定**
yolov4.cfg（設定模型組態）
obj.data（物件資料設定）
obj.names（物件類別名稱）

**資料內容**
*.jpg（測試影像）

# 修改Makefile並編譯

## 啟用OpenCV, GPU, CUDNN, CUDNN_HALF

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

!make # 進行編譯
```

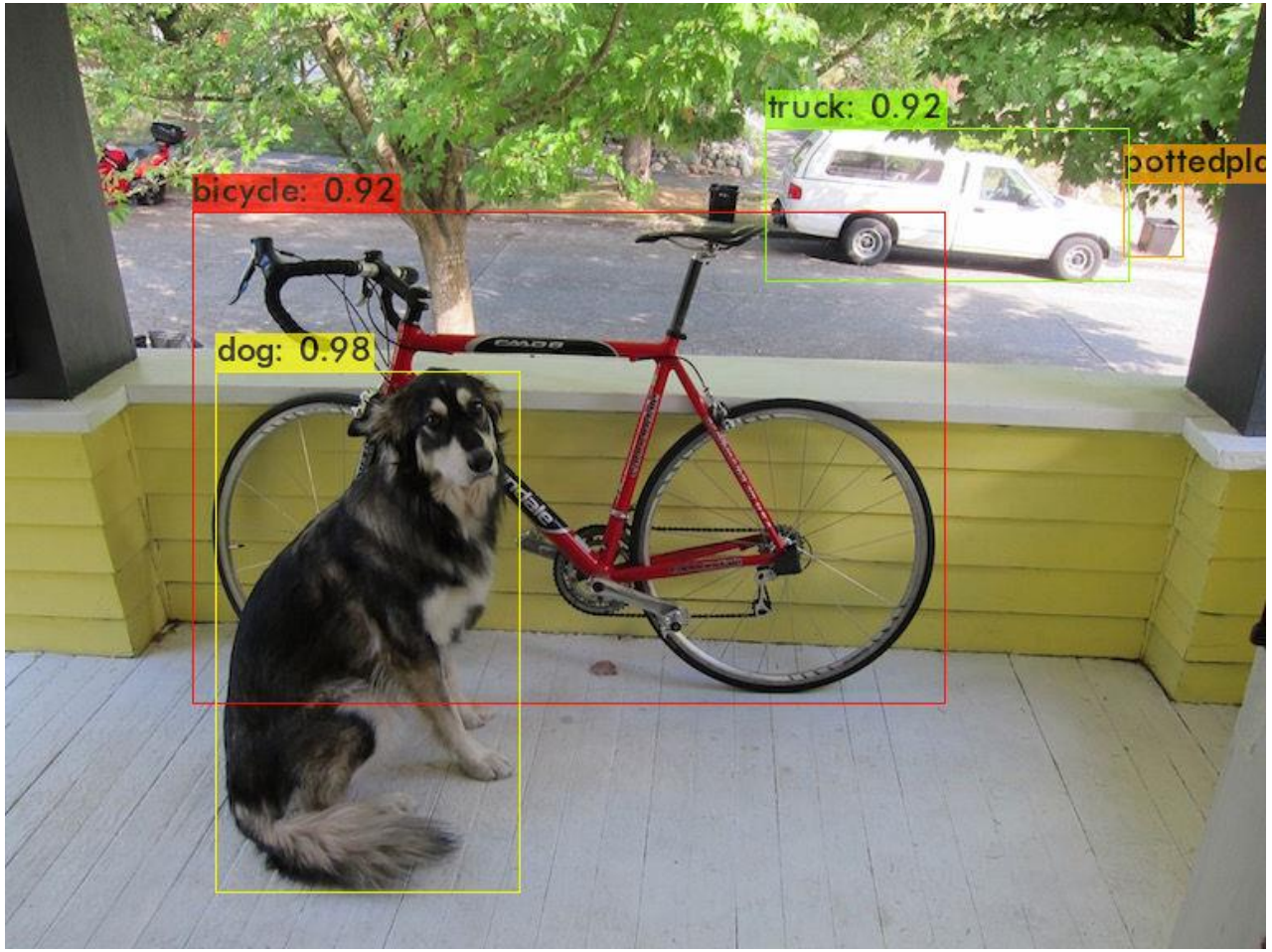
# 測試Yolov4 (darknet)

```
!./darknet detector test cfg/coco.data cfg/yolov4.cfg /my_drive/yolov4/yolov4.weights  data/dog.jpg
```

```
import cv2 # 導入OpenCV函式庫
from google.colab.patches import cv2_imshow # 導入Colab.patches函式庫
```

```
imgResult = cv2.imread('predictions.jpg') # 讀入結果影像
cv2_imshow(imgResult) # 顯示結果影像
```

# 預訓練模型測試結果




data/dog.jpg: Predicted in 165.845000 milli-seconds.

bicycle: 92%
dog: 98%
truck: 92%
pottedplant: 33%

# 自定義模型訓練

1. 準備資料集（影像檔）
2. 使用LabelImg標注工具製作符合YOLO格式
3. 將步驟3準備好的相關設定及預訓練檔從 Google Drive /yolov4 路徑下複製到Colab指定路徑下，包含下列六種檔案。
   2.1 obj.data （物件資料設定）
   2.2 obj.names （物件類別名稱）
   2.3 yolov4.cfg （設定模型組態）
   2.4 train.txt （訓練內容，另含原始影像壓縮檔）
   2.5 valid.txt （驗證內容，另含原始影像壓縮檔）
   2.6 pre-trained.weight （預訓練權重檔 yolov4.conv.137(yolov4), yolov4_conv.29(yolov4-tiny)）

https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.conv.137

# 自訂義資料集


img_001.jpg


img_002.jpg


img_003.jpg


img_004.jpg


img_005.jpg


img_006.jpg


img_010.jpg


img_011.jpg


img_012.jpg


img_013.jpg


img_014.jpg


img_015.jpg


img_019.jpg


img_020.jpg


img_021.jpg


img_022.jpg


img_023.jpg


img_024.jpg


img_028.jpg


img_029.jpg


img_030.jpg


img_031.jpg


img_032.jpg

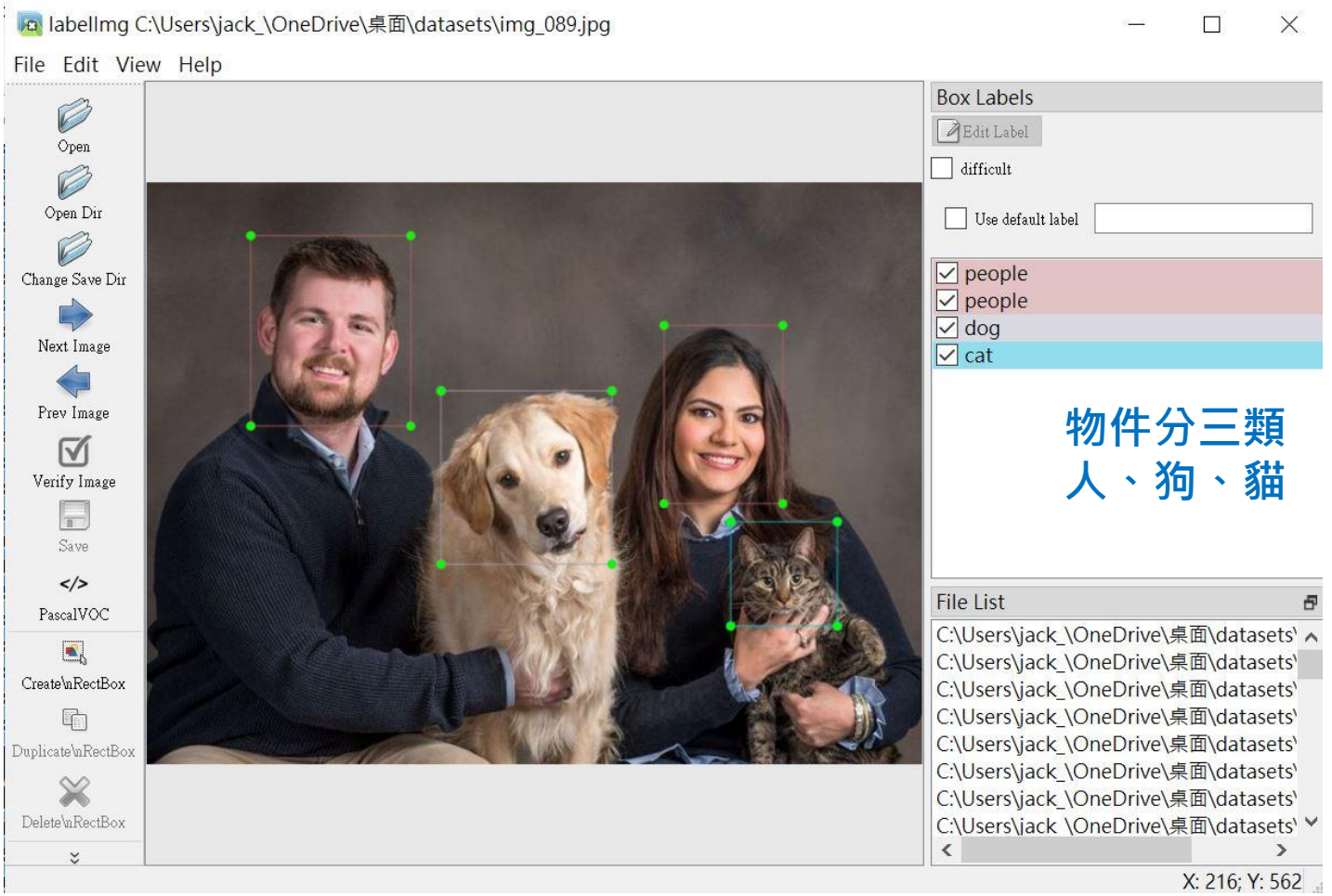

img_033.jpg

**100張影像，其中img_001-img_080作為訓練用，其餘20張作為驗證用。**

# LabelImg物件標註

預設標註檔為VOC格式，YOLO格式要用另存新檔。

選擇輸出格式VOC(*.xml) / YOLO(*.txt)




https://github.com/tzutalin/labelImg

# 建立自定義訓練相關檔案

- my_obj.data

classes = 3 **(類別數量）**
train = data/my_train.txt **（訓練集）**
valid = data/my_test.txt **（驗證集）**
names = data/my_obj.names **（類別名稱）**
backup = /my_drive/yolov4/ **（自動備份路徑）**

- my_obj.names

dog
cat
people

# 修改自訂義組態檔

- my_yolov4_custom.cfg

  (從/cfg/yolov4_custom.cfg複製而得 )

```
batch=64 # line 6
subdivisions=16 # line 7
width=416   # line 8, 32倍數
height=416  # line 9, 32倍數

max_batches = 6000 # line 20, classes*2000

steps=4800,5400  # line 22, max 80%, 90%
```

**有三處要修改**
**filter數量為(classes+5)*3**

```
[convolutional]
filters=24 # 第963, 1051, 1139列

[yolo]
classes=3 # 第970, 1058, 1146
```

# 執行自定義模型訓練

- 正常（重頭）執行

!./darknet detector train data/my_obj.data cfg/my_yolov4_custom.cfg yolov4.conv.137 -dont_show

**每1000次會產生一個暫存檔my_yolov4_custom_x000.weights ，最後會產生一個完成檔my_yolov4_custom_final.weights**

- 接續執行

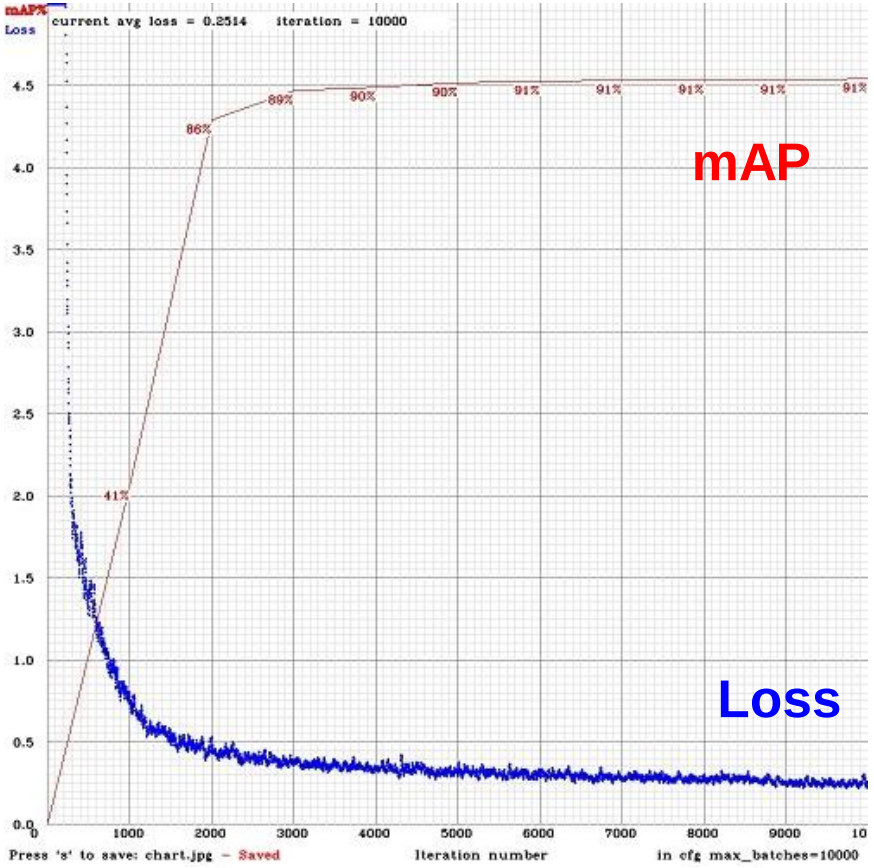
!cp /my_drive/yolov4/my_yolov4_custom_last.weights backup/

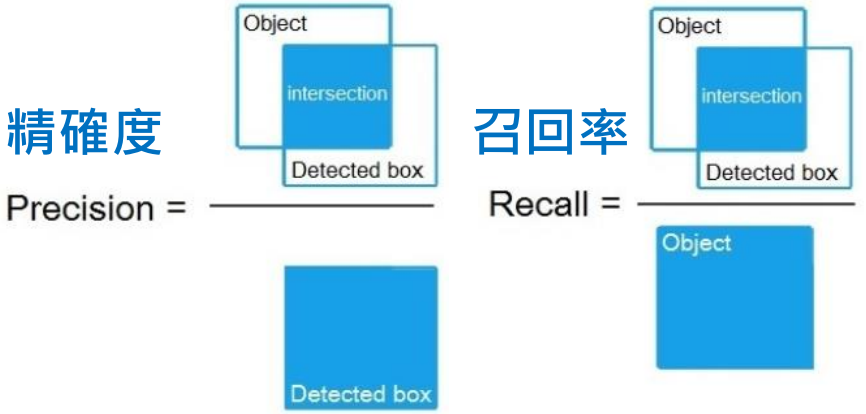
!ls backup/

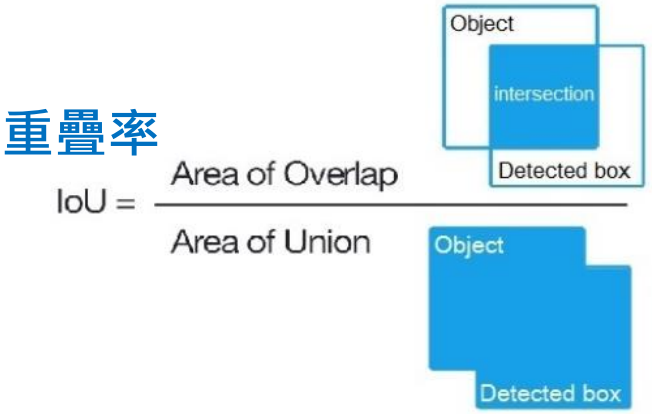
!./darknet detector train data/my_obj.data cfg/my_yolov4_custom.cfg backup/my_yolov4_custom_last.weights -dont_show

# 訓練損失Loss與平均精確度mAP

./darknet detector train data/obj.data yolo-obj.cfg yolov4.conv.137 -map



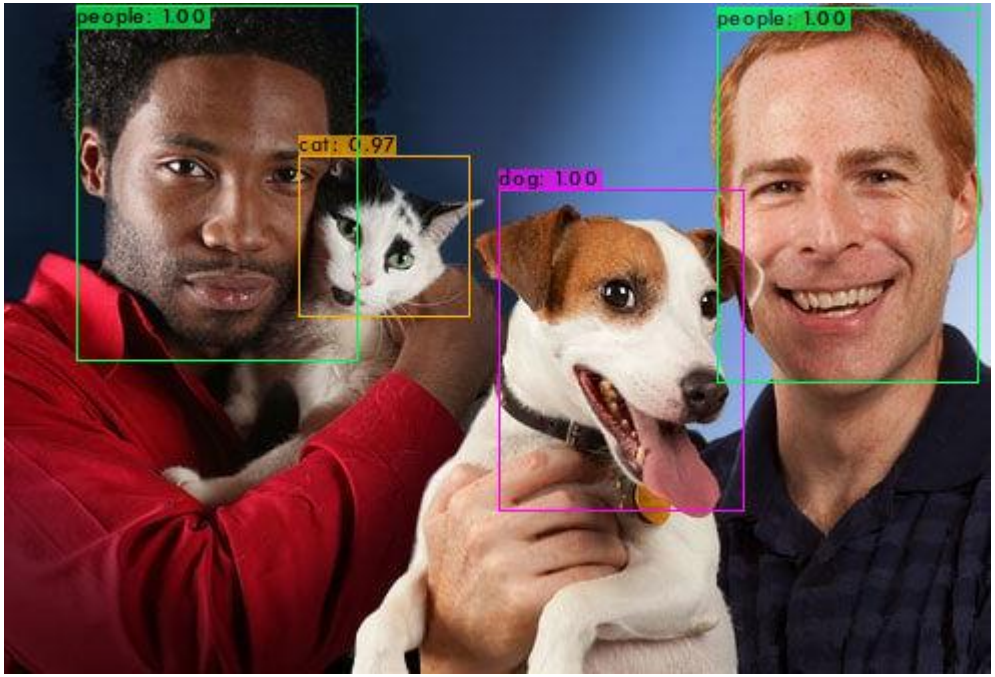




資料來源：https://github.com/AlexeyAB/darknet

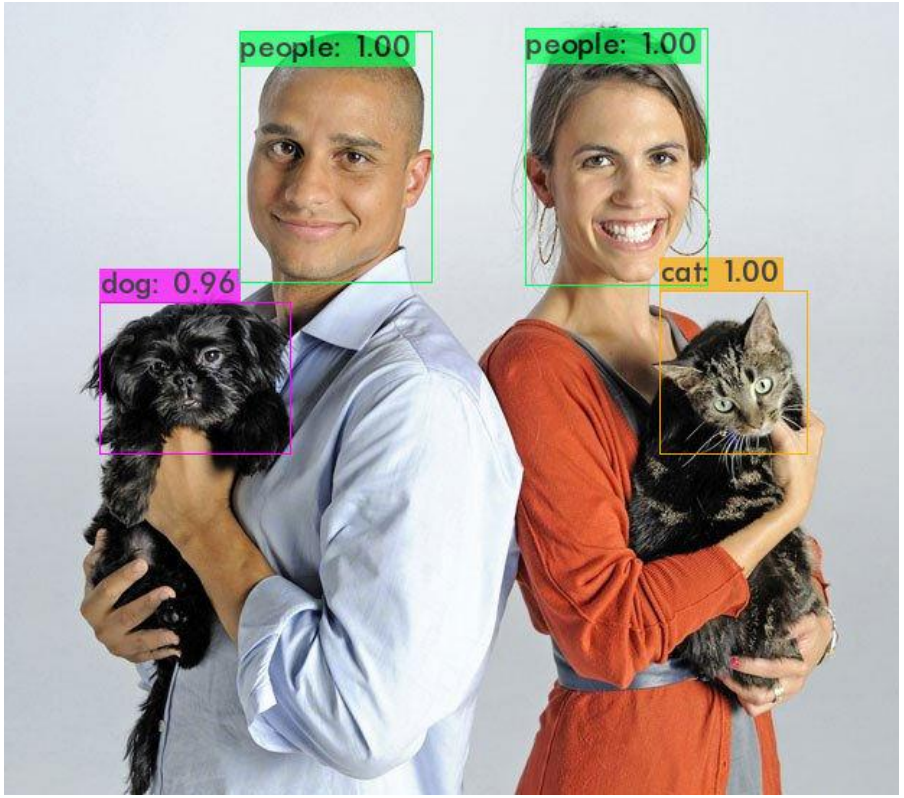
# 執行自定義模型推論

```
!./darknet detector test data/my_obj.data cfg/my_yolov4_custom.cfg backup/my_yolov4_custom_final.weights /my_drive/yolov4/test01.jpg


import cv2 # 導入OpenCV函式庫

from google.colab.patches import cv2_imshow # 導入Colab.patches函式庫


imgResult = cv2.imread('predictions.jpg') # 讀入結果影像

cv2_imshow(imgResult) # 顯示結果影像
```

# 自定義模型測試結果




test01.jpg




test02.jpg

# Yolov4 vs. Yolov4-tiny

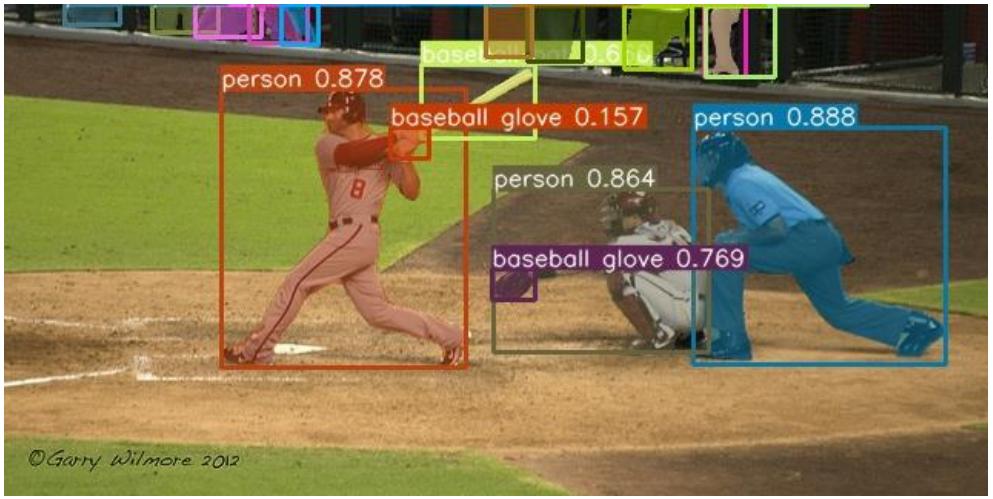
| | Yolov4 | Yolov4-tiny |
|---|---|---|
| 預訓練 Config | yolov4.cfg | yolov4-tiny.cfg |
| 預訓練 Weight | yolov4.weights (245 MB) | yolov4-tiny.weight (23.1 MB) |
| 自定義 Config | yolov4-custom.cfg | yolov4-tiny-custom.cfg |
| 自定義 Weight | yolov4.conv.137 (162 MB) | yolov4-tiny.conv.29 (19 MB) |
| 適用領域 | 雲端 / 桌機 | 單板電腦 / 小型AI晶片 |

參考資料：https://github.com/AlexeyAB/darknet#how-to-train-to-detect-your-custom-objects

# YOLOv7主要功能


物件偵測

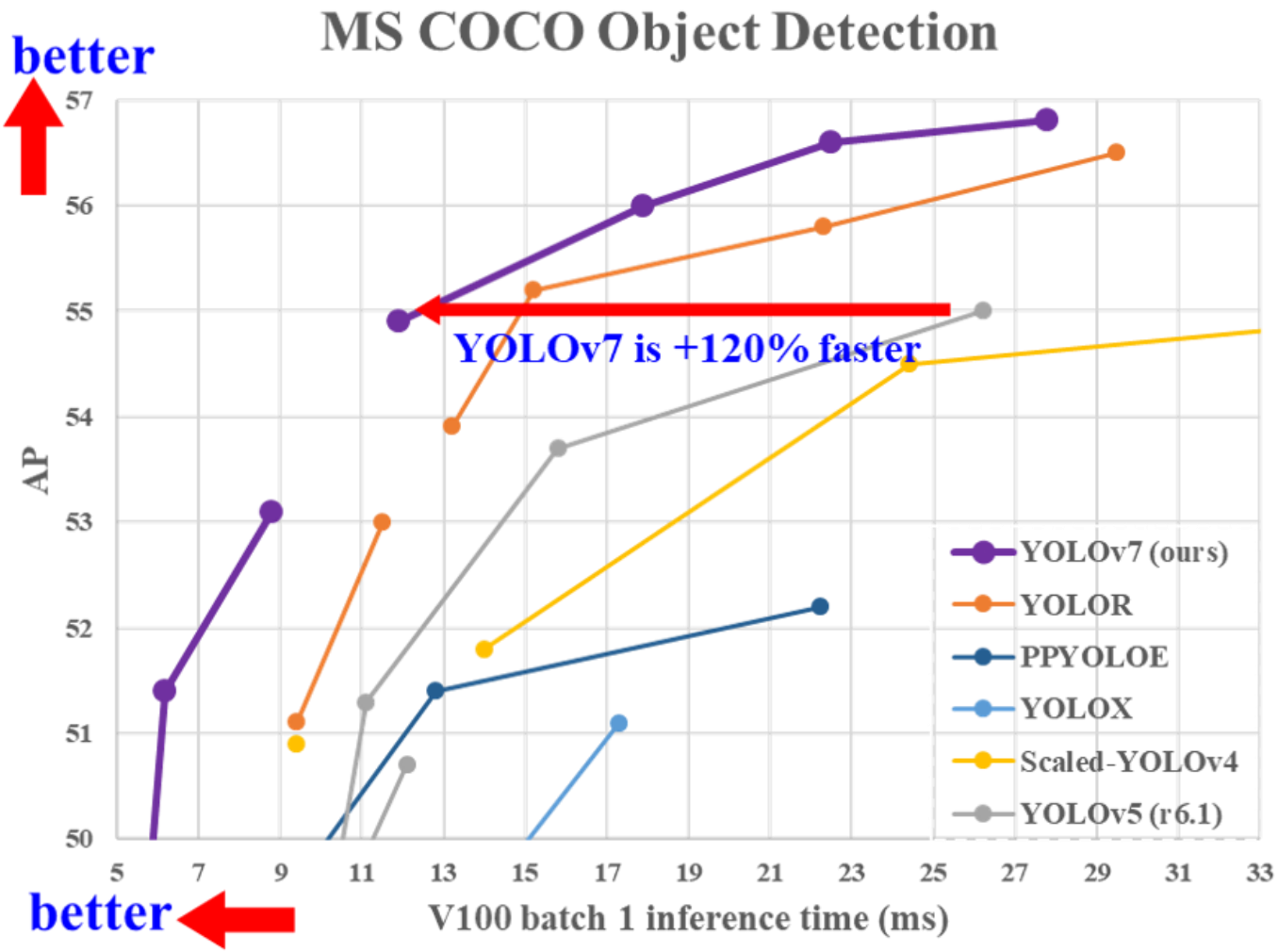
姿態估測




實例分割

https://github.com/WongKinYiu/yolov7

# YOLOv7延遲與精確度比較表




資料來源：https://github.com/WongKinYiu/yolov7

# YOLOv7效能比較

MS COCO

**Nvidia V100**

| Model | Test Size | $AP^{test}$ | $AP_{50}{}^{test}$ | $AP_{75}{}^{test}$ | batch 1 fps | batch 32 average time |
|---|---|---|---|---|---|---|
| YOLOv7 | 640 | 51.4% | 69.7% | 55.9% | 161 *fps* | 2.8 *ms* |
| YOLOv7-X | 640 | 53.1% | 71.2% | 57.8% | 114 *fps* | 4.3 *ms* |
| | | | | | | |
| YOLOv7-W6 | 1280 | 54.9% | 72.6% | 60.1% | 84 *fps* | 7.6 *ms* |
| YOLOv7-E6 | 1280 | 56.0% | 73.5% | 61.2% | 56 *fps* | 12.3 *ms* |
| YOLOv7-D6 | 1280 | 56.6% | 74.0% | 61.8% | 44 *fps* | 15.0 *ms* |
| YOLOv7-E6E | 1280 | 56.8% | 74.4% | 62.1% | 36 *fps* | 18.7 *ms* |

資料來源：https://github.com/WongKinYiu/yolov7

# Ex4: OpenVINO YOLOv7實作 (1/3)

1. 確定已進入python虛擬環境openvino_env中且已安裝好openvino-dev[ONNX,tensorflow2,pytorch]==2022.1.0 。
2. 下載Yolov7並進入工作路徑中。

```
cd /
git clone https://github.com/WongKinYiu/yolov7.git
cd yolov7
```

3. 下載Yolov7預訓練好的權重到yolov7路徑下，如yolov7-x, yolov7-w6等，這裡使用最小的 yolov7.pt作為練習。
   https://github.com/WongKinYiu/yolov7/releases/download/v0.1/yolov7.pt
4. 安裝yolov7所需套件

```
pip install -r requirements.txt
```

# Ex4: OpenVINO YOLOv7實作 (2/3)

5. 確認yolov7是否已可正常工作。
   - 使用Webcam，python detect.py --weight yolov7.pt --source 0
   - 使用影像檔，python detect.py --weights yolov7.pt --conf 0.25 --img-size 640 --source ./inference/images/horses.jpg
   - 產生的結果會出現在./runs/detect/expn下，n表示第幾次產出的結果
6. 將pt檔轉成ONNX格式，**python export.py --weight yolov7.pt**
7. 執行效能分析以取得模型輸出入資訊
   **benchmark_app -m yolov7.onnx**

```
[Step 6/11] Configuring input of the model
[ INFO ] Model input 'images' precision u8, dimensions ([N,C,H,W]): 1 3 640 640
[ INFO ] Model output 'output' precision f32, dimensions ([...]): 1 3 80 80 85
[ INFO ] Model output '528' precision f32, dimensions ([...]): 1 3 40 40 85
[ INFO ] Model output '548' precision f32, dimensions ([...]): 1 3 20 20 85
```

# Ex4: OpenVINO YOLOv7實作 (3/3)

8. 將ONNX格式轉成OpenVINO IR格式(xml, bin)

```
mo --input_model yolov7.onnx --input images --output output,528,548 --data_type FP16
```

9. 下載OpenVINO Yolov7推論範例，並進到工作路徑下。

```
cd /
git clone https://github.com/OpenVINO-dev-contest/YOLOv7_OpenVINO_cpp-python.git
cd YOLOv7_OpenVINO_cpp-python/python/
```

10. 安裝必要套件包。

```
pip install -r requirements.txt
```

# OpenVINO YOLOv7推論比較 (1/2)

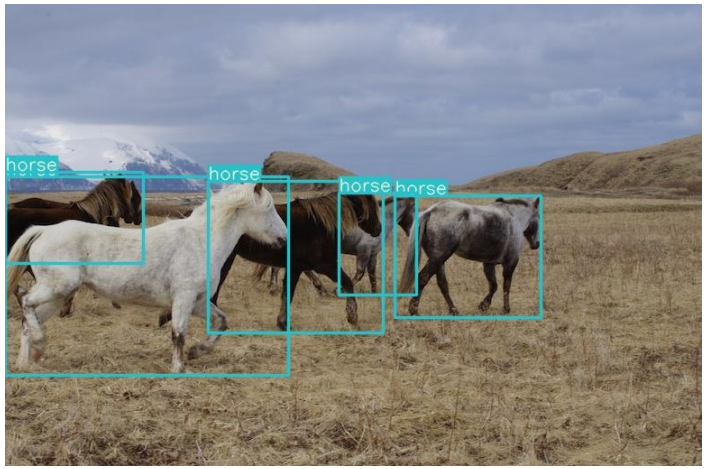
1. 進入OpenVINO Yolov7範例路徑下

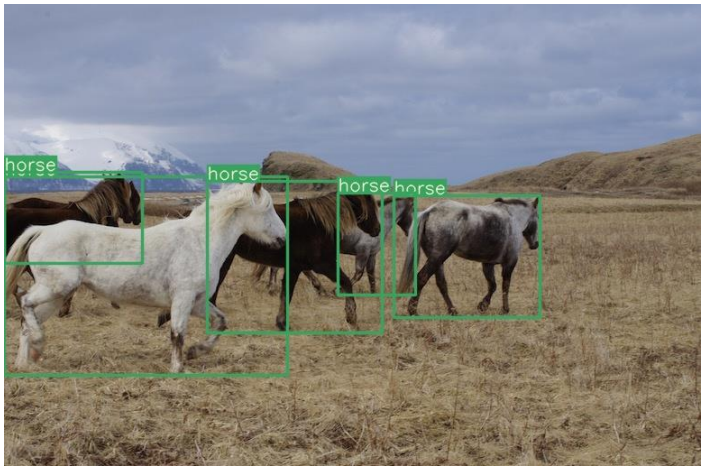
**cd YOLOv7_OpenVINO_cpp-python/python/**

2. 分別使用ONNX和IR格式進行推論，確認辨識差異，幾乎沒差。

**python image.py -m X:\yolov7\yolov7.onnx -i ..\data\horses.jpg**

**python image.py -m X:\yolov7\yolov7.xml -i ..\data\horses.jpg**



# OpenVINO YOLOv7推論比較 (2/2)

3. 重新回到yolov7路徑下，執行benchmark_app比較運行效率。

**cd x:\yolov7**

**benchmark_app -m yolov7.onnx -t 15 -d [裝置]**

**benchmark_app -m yolov7.xml -t 15 -d [裝置]**

CPU
GPU
MULTI：CPU,GPU

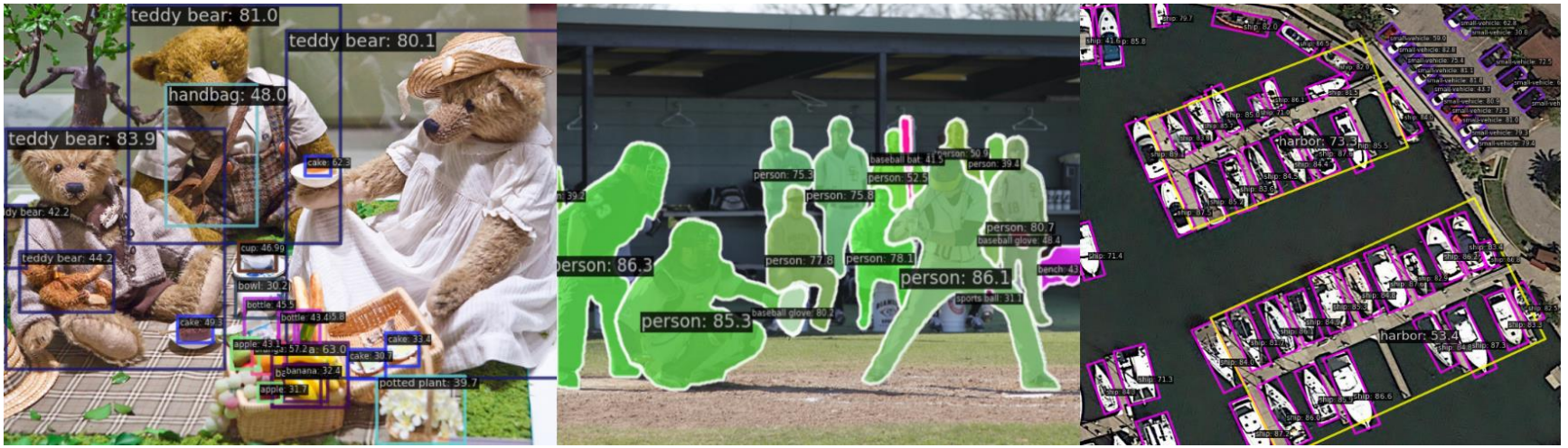
4. 經比較後可看出經過OpenVINO優化後的模型運行效率都有提高，甚至在iGPU快了將近90%。

| [裝置] | CPU（i7-9750) | iGPU (UHD630) | MULTI:CPU,GPU |
|---|---|---|---|
| ONNX | 2.92 FPS | 2.61 FPS | 3.96 FPS |
| IR(XML,BIN) | 3.07 FPS | 4.96 FPS | 6.33 FPS |
| 速度提升 | 1.05x | 1.90x | 1.59x |

註：運行效率僅供參考，在不同模型及測試條件下可能會產生不一樣的結果。

# 延伸閱讀：旋轉物件偵測

MMRotate https://github.com/open-mmlab/mmrotate





**空拍影像及旋轉物件資料集（DOTA）**

https://captain-whu.github.io/DOTA/

# 參考文獻


- pyimagesearch - Introduction to the YOLO Family

https://pyimagesearch.com/2022/04/04/introduction-to-the-yolo-family/

- YOLO Github

https://pjreddie.com/darknet/yolo/

- Darknet Github

https://github.com/AlexeyAB/darknet

- 完整範例程式

https://github.com/OmniXRI/Yolov4_Colab_User_Datasets (yolov4)

https://github.com/OmniXRI/Yolov4-tiny_Colab_User_Datasets (yolov4-tiny)

- 許哲豪，如何以Google Colab及Yolov4-tiny來訓練自定義資料集─以狗臉、貓臉、人臉偵測為例

https://omnixri.blogspot.com/2021/05/google-colabyolov4-tiny.html