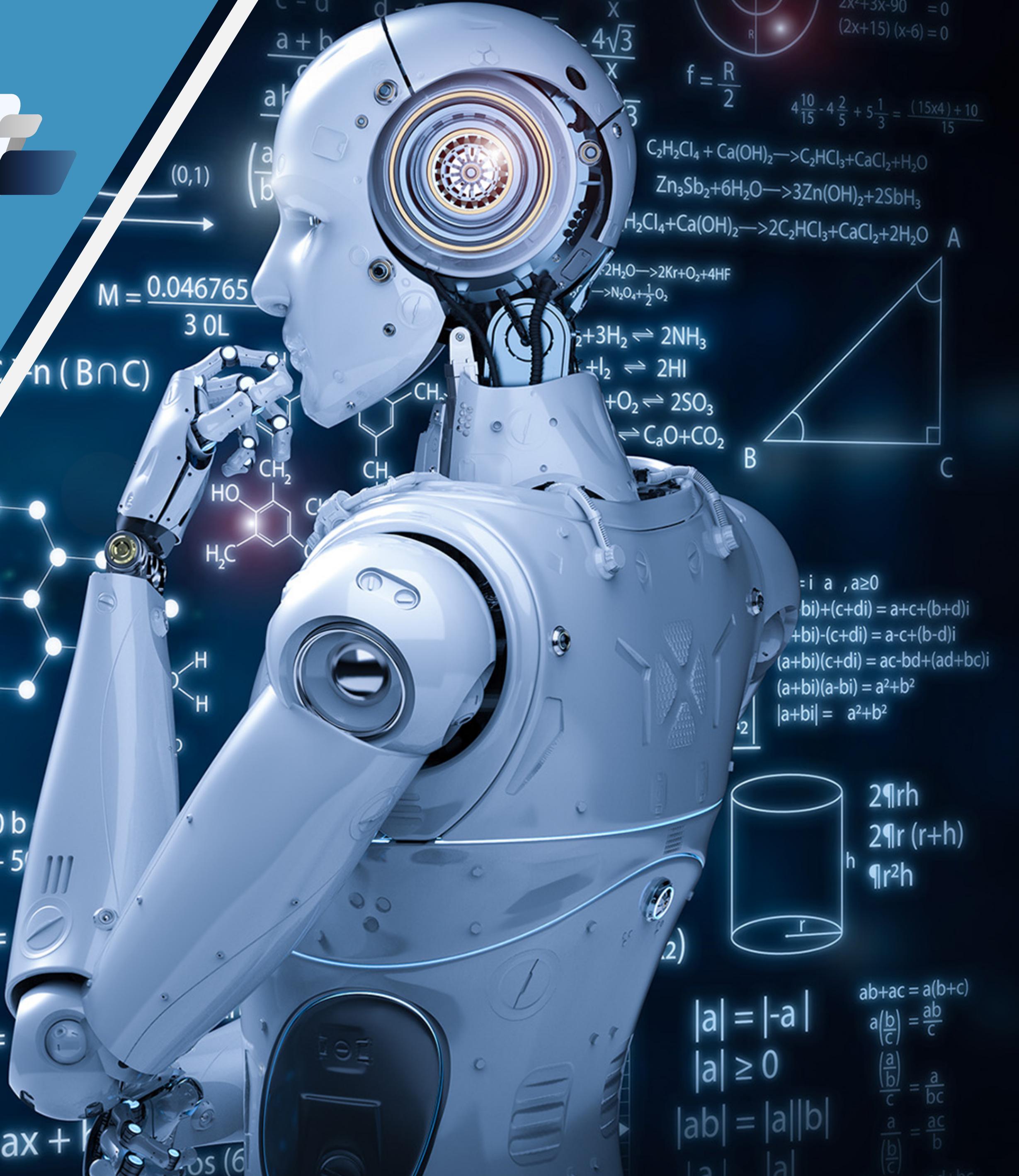


# Day 40

## 深度學習與電腦視覺 學習馬拉松

cupay 陪跑專家：杜靖愷



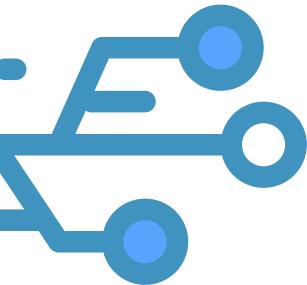


更快的檢測模型 - tiny YOLOv3



# 重要知識點

- 如何評估檢測模型的執行速度。
- 如何調用 tiny YOLOv3。



# tinyYOLOv3 簡介



```
def tiny_yolo_body(inputs, num_anchors, num_classes):
    '''Create Tiny YOLO_v3 model CNN body in keras.'''
    x1 = compose(
        DarknetConv2D_BN_Leaky(16, (3,3)),
        MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='same'),
        DarknetConv2D_BN_Leaky(32, (3,3)),
        MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='same'),
        DarknetConv2D_BN_Leaky(64, (3,3)),
        MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='same'),
        DarknetConv2D_BN_Leaky(128, (3,3)),
        MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='same'),
        DarknetConv2D_BN_Leaky(256, (3,3)))(inputs)

    x2 = compose(
        MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='same'),
        DarknetConv2D_BN_Leaky(512, (3,3)),
        MaxPooling2D(pool_size=(2,2), strides=(1,1), padding='same'),
        DarknetConv2D_BN_Leaky(1024, (3,3)),
        DarknetConv2D_BN_Leaky(256, (1,1)))(x1)

    y1 = compose(
        DarknetConv2D_BN_Leaky(512, (3,3)),
        DarknetConv2D(num_anchors*(num_classes+5), (1,1)))(x2)

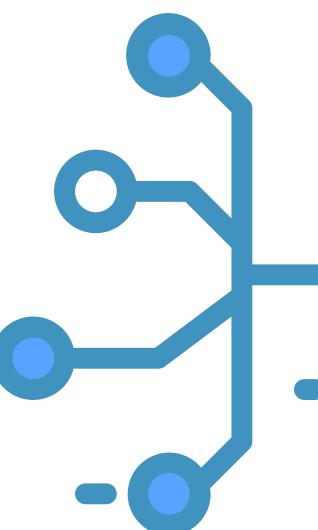
    x2 = compose(
        DarknetConv2D_BN_Leaky(128, (1,1)),
        UpSampling2D(2))(x2)

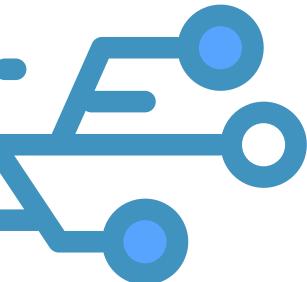
    y2 = compose(
        Concatenate(),
        DarknetConv2D_BN_Leaky(256, (3,3)),
        DarknetConv2D(num_anchors*(num_classes+5), (1,1)))([x2,x1])

    return Model(inputs, [y1,y2])
```

在計算資源有限的情況下，作者有提出另一個更輕量的模型 - tinyYOLOv3，相比 Darknet-53 就沒有那麼深；除此之外 YOLOv3 輸出 3 個 scale 的 feature map，tinyYOLOv3 只輸出 2 個 scale。具體細節可以直接參考[程式碼中的實現](#)。

模型最終輸出兩個不同 scale 的 feature map





# 如何評估及計算檢測模型的速度？

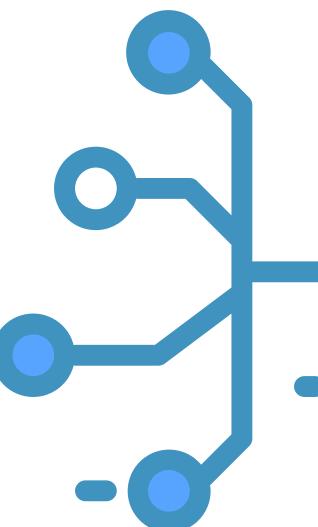


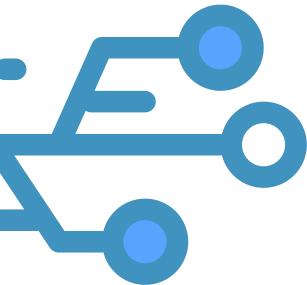
最直接的方法就是去衡量檢測模型在一張圖像上執行的時間

```
import time  
  
start = time.time()  
results = yolo.pure_detect_image(image)  
end = time.time()  
print("single inference took: %.3fs" % (end-start))
```

由於原來的 `detect_image` 包含了繪圖的程式碼，程式碼範例中另外定義了一個單純只有網路檢測的函數，以便精準估計檢測模型執行的時間。

single inference took: 0.162s

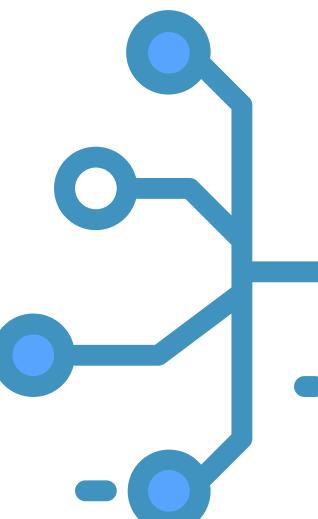


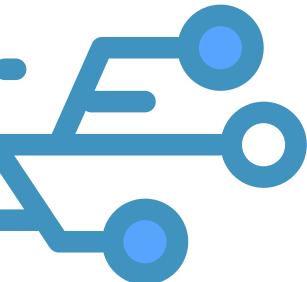


# 如何評估及計算檢測模型的速度？



應用場景中，常常需要檢測模型能夠實時運行，那什麼叫實時呢？相比模型執行單次的時間，我們會去看另一個叫 Frame Per Second (FPS) 的指標，也就是一秒能跑的圖片數量。一般電影是 24 FPS，而 YOLO 使用 Titan X 能達到 45 FPS。





# 如何評估及計算檢測模型的速度？

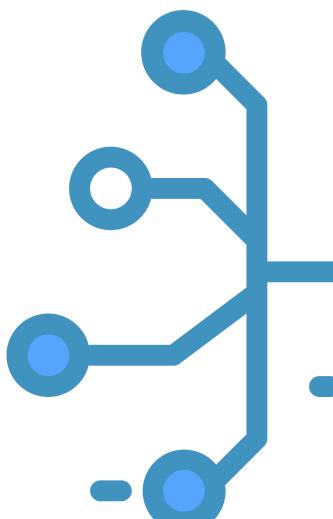


FPS 的計算也很簡單，單次執行時間的倒數即是 FPS

```
fps = 1 / (end-start)  
print("fps: %.3f" % (fps))
```

fps: 6.166

Colab K80 GPU



# 專家筆記

第一次執行檢測網路的時間會比較久，因為需要花時間配置 GPU 記憶體。

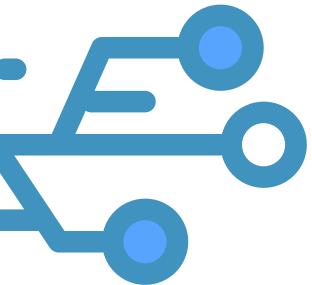
可參考以下連結：

<https://github.com/keras-team/keras/issues/8724>



# 知識點 回顧

- 用 FPS 來評估檢測模型的實時性，也就是一秒能檢測多少張圖片，具體多少就看實際場景的需求
- 同一個模型在不同 GPU 上有不同的執行時間
- 沒有白吃的午餐，比較快的模型在精準度方面必定有犧牲，是 tradeoff



# 參考資料

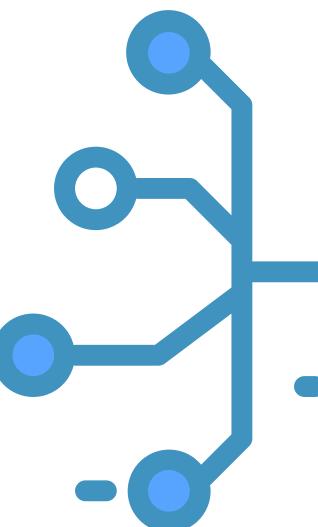
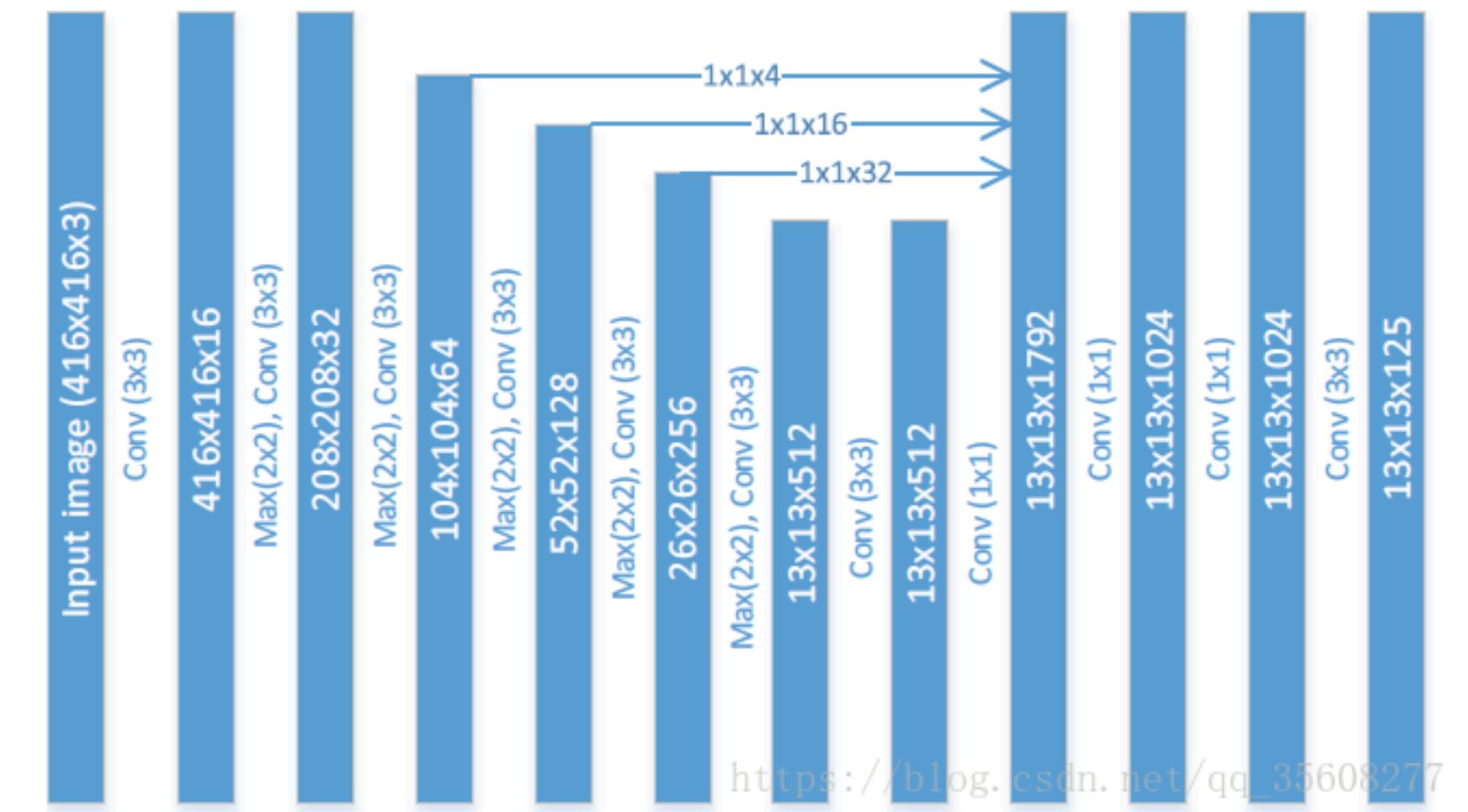


- 我們的肉眼每秒能夠捕捉多少畫面？
- tiny-yolov3 的網路結構圖

## yolov3-tiny的训练

原创 张学渣 发布于2018-12-23 14:38:55 阅读数 10175 ☆ 收藏

yolov3-tiny



# 解題時間 Let's Crack It



請跳出 PDF 至官網 Sample Code & 作業開始解題