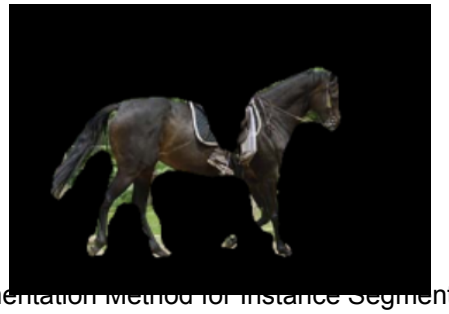
I. Data Preprocessing

- **Image Preprocessing**
  - random_translate: 對原本圖片進行平移。
  - random_crop: 對原本圖片進行裁切。
  - random_horizontal_flip: 對原本圖片做水平翻轉。

- **Copy-Paste Augmentation**

  參考*"Simple Copy-Paste is a Strong Data Augmentation Method for Instance Segmentation"*[1]，該篇論文主要是呈現若將物件圖片做複製黏貼產生新的圖片作為data可以大大增加object detection的準確率，故我們決定實作此方法:首先將原本的training set通過mask RCNN(網路上用MS coco dataset training過的model[2])，偵測每張圖片的Object並將其mask透過pikle檔存起來，mask會是bool type的圖片陣列，有了這些圖片陣列我們可以將其做成一張張貼紙，如下圖(左邊是原圖，中間是mask，右邊是原圖加上mask):

每張貼紙由"原圖位址+mask+類別"組成，並將貼紙分類存成Dictionary，keys是類別，value是貼紙組成的list。因為Mask RCNN產生的mask並不是每張都很準確，如下圖的馬被切開，或是很多小物件被偵測成錯誤的類別，所以我們人工label了我們要拿來copy paste的貼紙。



[1] Simple Copy-Paste is a Strong Data Augmentation Method for Instance Segmentation
Golnaz Ghiasi, Yin Cui, Aravind Srinivas, Rui Qian, Tsung-Yi Lin, Ekin D. Cubuk, Quoc V. Le, Barret Zoph, *13 Dec 2020*

[2] https://github.com/matterport/Mask_RCNN

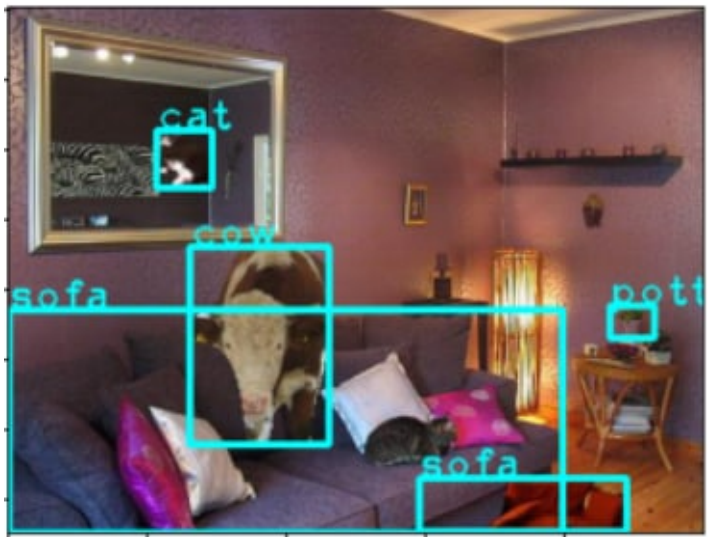
接著，我們重複下列動作，不斷產生新的訓練資料：

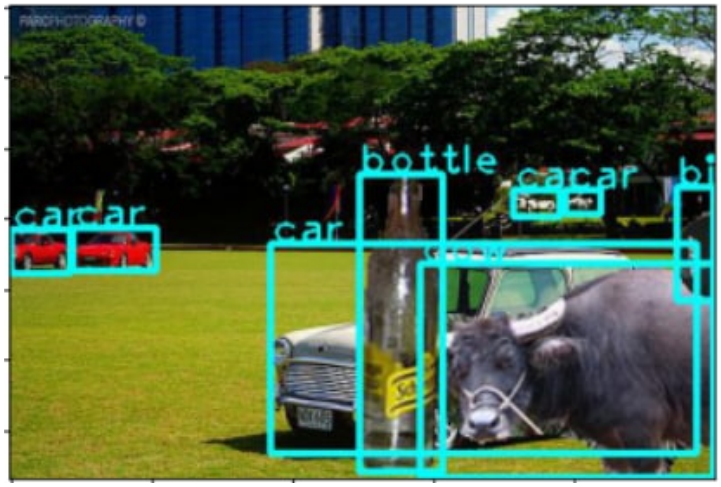
1. 以根據訓練資料中物件各類別數量分布的倒數取出一張圖為背景圖，如:目前資料集"牛"的物件較少，"人"的物件較多，故取出有"牛"的原圖的機率會比取出有"人"的原圖機率還要高
2. 隨機選擇要貼上的貼紙數(1~3張)
3. 根據2的次數，隨機抽取數張貼紙(如1機率分布的選擇方式)平移、縮放後貼在背景圖上
4. 儲存貼完貼紙後的新圖，新貼上的物件bounding box即為貼紙mask的bounding box，回到步驟1。

上面的步驟我們在code裡排除了一些問題:

1. 縮放後的貼紙可能會過大，佔滿整張圖，所以設一個貼紙面積的gate(不能超過原圖面積的40%)
2. 貼紙可能會覆蓋掉圖上的其他物件，若覆蓋其他物件的70%以上則重貼

最後產生出來的圖如下例：





II. What kind of models you have tried and how did they work

- YOLOv1 with Pretrained TF Application
  - ResNet152v2 weight ImageNet
    - get layer: conv4_block36_3_conv (Conv2D) (None, 14, 14 1024)
    - 我們將上述layer的ouput作為YOLOv1的input，接著做MaxPooling和conv2+LeakyReLU，分辨以及框線效果相當不理想。

  - ResNet101v2 weight ImageNet
    - get layer: conv4_block23_3_conv (Conv2D) (None, 14, 14, 1024)
    - 我們將上述layer的ouput作為YOLOv1的input，接著做MaxPooling和conv2+LeakyReLU。分辨以及框線效果也相當不理想。因此萌生了我們重新建構model的想法。

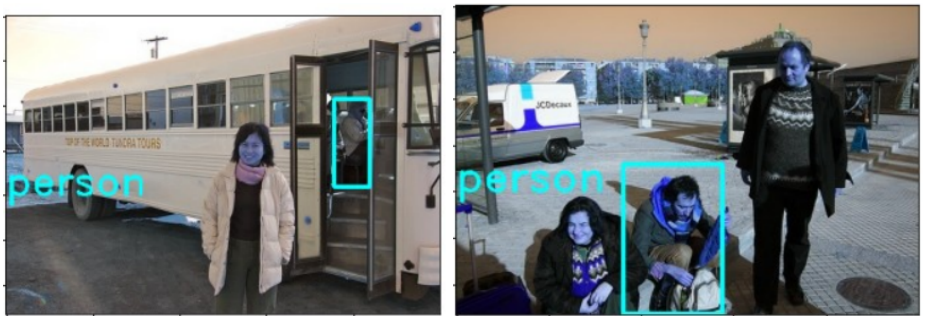
- YOLOv4 on Original Dataset

  我們將我們寫好的 yolov4 model 訓練在這次競賽的原始資料集上，我們將模型訓練至收斂後，我們發現 evaluation 出來的結果比使用 Copy-Paste Augmentation 資料集進行訓練的模型還差，因此我們就終止使用 Original Dataset 進行模型訓練，轉而使用我們所生產出的 Copy-Paste Augmentation 資料集，也肯定 Copy-Paste Augmentation 對效能提升的重要性。

- YOLOv4 on Copy-Paste Augmentation

  基於Copy-Paste資料集，我們train YOLOv4直到它epoch之間的Loss差降到0.02，大概耗費兩天的時間。而後來實驗發現，當Loss差距在0.5之間，其實performance差距很小，以 evaluation出來的結果而言，大概會相差0.01。

  以下是以圖片中YOLOv4取出conf最高的當作bounding box output，可以看到YOLOv4在圖片細微的地方也能有很高的框框精準度。



- Transfer Learning

  在了解到 Copy-Paste Augmentation 的重要性後，我們使用這一項方法，來產生更多的資料對模型進行訓練。我們先後產生 cpaug-1, cpaug-2, cpaug3 這三份資料集，結尾尾數越大的資料集，在單一圖片所貼上的物件較多。

  我們一開始只有在 cpaug-1 上進行訓練的模型，且收斂時間需要兩天，因此我們預期若是從頭使用 cpaug-2 訓練的話，會需要超過兩天的時間，因為資料集的難度較高。於是我們將在 cpaug-1 上進行訓練的模型遷移至 cpaug-2 進行訓練。除此之外，我們也有嘗試把遷移至 cpaug-2 進行訓練的模型再遷至難度更高的 cpaug3

  - Transfer From cpaug-1 to cpaug-2

    遷移的過程相當順利，訓練初始的 loss 並沒有跳到很高，模型也有穩定的收斂，幫助我們節省很多訓練時間。在模型收斂過後我們自行計算 mse，得到的分數是 0.33，相比只在 cpaug-1 進行訓練的模型低了 0.03，因此我們認為這次的遷移的確有對效能造成提升。

  - Transfer From cpaug-2 to cpaug-3

有了前面的經驗，加上更多的人工標註資料，我們產生了難度更高的 cpaug-3 並將前面訓練在 cpaug-2 模型轉移到 cpaug-3。不過很遺憾的是我們的模型並沒有完全收斂，自行計算 mse 後得到的分數是 0.336，效能與訓練在 cpaug-2 模型相當，卻在 Private Score Board 拿了較高的成績。因此我們相信我們還能突破自己的最佳成績，只是礙於時間限制而無法達成。

- Fine-Tuning

  模型使用 cpaug-2 訓練達到收斂後，我們嘗試調高 batch size，預期能讓模型的 loss 降低，然而因為訓練時間不夠，模型無法達到收斂，最後自行計算 mse 後得到 0.3422，比原本遷移過來的模型還差。

	mse
cpaug-1	0.357
cpaug-2	0.33
cpaug-3	0.336
cpaug-2 increase batch size	0.342

III. What problems occurred and how did you solve them

- Data Imbalance

  Original Dataset的數量相當不平均，由下圖可以見到person的數量高出其他class甚至十倍，這導致初期訓練的model預測結果都趨向於預測class 14 (person)，最終MSE也都高於0.90。

Table 1: Statistics of the main image sets. Object statistics list only the `non-difficult' objects used in the evaluation.

	train		val		trainval		test	
	img	**obj**	**img**	**obj**	**img**	**obj**	**img**	**obj**
Aeroplane	112	151	126	155	238	306	-	-
Bicycle	116	176	127	177	243	353	-	-
Bird	180	243	150	243	330	486	-	-
Boat	81	140	100	150	181	290	-	-
Bottle	139	253	105	252	244	505	-	-
Bus	97	115	89	114	186	229	-	-
Car	376	625	337	625	713	1250	-	-
Cat	163	186	174	190	337	376	-	-
Chair	224	400	221	398	445	798	-	-
Cow	69	136	72	123	141	259	-	-
Diningtable	97	103	103	112	200	215	-	-
Dog	203	253	218	257	421	510	-	-
Horse	139	182	148	180	287	362	-	-
Motorbike	120	167	125	172	245	339	-	-
Person	1025	2358	983	2332	2008	4690	-	-
Pottedplant	133	248	112	266	245	514	-	-
Sheep	48	130	48	127	96	257	-	-
Sofa	111	124	118	124	229	248	-	-
Train	127	145	134	152	261	297	-	-
Tvmonitor	128	166	128	158	256	324	-	-
Total	2501	6301	2510	6307	5011	12608	-	-

- 調整class weight
  我們嘗試依據各種class的圖片數量去調整loss的懲罰比例，利用data數量的反比計算套入weighted cross entropy函式，其中第一項ground truth的代入就是p，第二項預測的結果就是p_P。
  數量反比的數列: class_weights = tf.constant([41.2026, 35.7167, 25.9424, 43.4759, 24.9663, 55.0568, 10.0864, 33.5319, 15.7995, 48.6795, 58.6419, 24.7216, 34.8287, 37.1917, 2.6883, 24.5292, 49.0584, 50.8387, 42.4512, 38.9136])
  代入計算的函式:
  tf.reduce_mean(tf.nn.weighted_cross_entropy_with_logits(
  tf.reshape(object_mask, (CELL_SIZE, CELL_SIZE, 1))*P,
  tf.reshape(object_mask, (CELL_SIZE, CELL_SIZE, 1))*p_P,
  pos_weight = cl_weights) * CLASS_SCALE)
  這個方法並沒有對整體結果帶來好的影響，反而會因為比例調整過大，專注於反比數列中非常大的幾個classes，而預測出person的結果也消失不見。

- Dataset擴充和數量調整
  對original dataset進行針對class數量的augmentation是最直接解決data imbalance的方法，但是初始數據集中的圖片幾乎不以單獨一個class標記的方式呈現，尤其大部分都混雜著person，所以很難以圖片倍增的方式處理數據不平衡。於是在執行Copy-paste augmentation時，便針對class數量去生成倒數比例數量的圖片。像是貼上的"preson”數量就少了許多，較少的"sheep"以及"cow"的數量就多很多，這樣我們訓練用的數據集就達到了相當程度的平衡。

- Multiple Prediction
  我們原本輸出在test_prediction.txt檔上的預測結果是單獨最有信心的那項預測，但是evaluate的結果都相當悽慘，尤其有相當多的0出現。之後，我們將model predict的code判斷最大confidence的部分取消掉，並在輸出於test_prediction.txt上的結果設一個信心大於0.3的threshold，最後就能在每張圖片都產生出許多confidence相當高的prediction。如此，evaluate出的數值就提升了許多。