VST Lab2 Report 多工碩一 313553024 蘇柏叡

1. Experiment Setup

1.0 環境建置之問題與解決

首先，參考[1][2]文章按部就班地下載、安裝相關套件，然而考量實作過程中轉檔後遇到讀不到annotations的問題，所以參考建議便嘗試將train、val之檔名設定為預設的train2017、val2017，並且將圖片放置到對應的資料夾之後即成功讀取(因為後續讀檔無問題，因此就持續使用train2017、val_2017作為資料夾名稱。

接著，在建立好環境、修改相關路徑之後，遇到了一個問題就是**執行setup.py後在yolox_base.py所修改之超參數設定無法在訓練時呈現**(比方說我調整lr、batch size但執行時仍顯示原先default)，因此我uninstall整個yolox之後，再重新下載後便可執行。

最後，我在eval驗證集時，由於我的環境一直無法使用FastCOCO(因source code無法用FastCOCO會exit)，因此我修改了jit_ops.py、fast_coco_eval_api.py這兩個檔案如圖1至圖5，使得當無法使用FastCOCO時轉回使用Regular COCO的評估方式。以此**解決因為找不到適合的C++編譯器、安裝C++編譯器後路徑問題解決不了而無法使用的問題**。

```
183         p = self.params
184         # Check if module was loaded successfully, else use standard evaluation
185         if not self.module:
186             print("FastCOCOEvalOp could not be loaded, using standard COCO evaluation.")
187             return super().evaluate()  # Fall back to standard COCOeval if fast_cocoeval is not available
```

```
260     def accumulate(self):
261         """
262         Accumulate per image evaluation results and store the result in self.eval.
263         """
264         if not self.module:
265             print("FastCOCOEvalOp could not be loaded, using standard COCO evaluation accumulate.")
266             return super().accumulate()  # Fall back to standard accumulate
267
```

圖1、圖2 fast_coco_eval_api.py 修改段落

```
208     def sources(self) -> List:
209         """Get path list of source files of op."""
210         try:
211             # 優先嘗試相對路徑
212             sources = glob.glob(os.path.join("yolox", "layers", "cocoeval", "*.cpp"))
213             if not sources:
214                 # 如果相對路徑找不到文件，嘗試絕對路徑
215                 import yolox
216                 code_path = os.path.join(yolox.__path__[0], "layers", "cocoeval", "*.cpp")
217                 sources = glob.glob(code_path)
218
219             if not sources:
220                 raise FileNotFoundError(f"Source files for {self.name} not found. Check your paths.")
221             return sources
222         except Exception as e:
223             print(f"Error finding source files for {self.name}: {e}")
224             return []  # 如果文件找不到，返回空列表以避免崩潰
```

```python
    def load(self, verbose=True):
        try:
            # 嘗試從已安裝的包中加載模組
            return importlib.import_module(self.absolute_name())
        except ModuleNotFoundError as e:
            print(f"Warning: {self.name} cannot be loaded due to {e}, using regular COCOeval instead.")
            return None  # 如果無法加載，返回 None
        except Exception as e:  # 捕獲其他異常情況
            print(f"An error occurred while loading {self.name}: {e}")
            return None  # 返回 None 或處理邏輯

    def jit_load(self, verbose=True):
        from torch.utils.cpp_extension import load
        from loguru import logger
        try:
            import ninja  # 檢查 Ninja 是否已安裝
```

```python
            build_tik = time.time()

            try:
                # 編譯 op 並加載
                op_module = load(
                    name=self.name,
                    sources=self.sources(),
                    extra_cflags=self.cxx_args(),
                    extra_cuda_cflags=self.nvcc_args(),
                    verbose=verbose,
                )
                build_duration = time.time() - build_tik
                if verbose:
                    logger.info(f"Load {self.name} op in {build_duration:.3f}s.")
                return op_module
            except Exception as e:
                logger.error(f"Failed to build {self.name} due to {e}")
                print(f"Compilation error: {e}. Falling back to regular COCOeval.")
                return None  # 返回 None 避免崩潰

        def clear_dynamic_library(self):
            """Remove dynamic libraray files generated by JIT compilation."""
            module = self.load()
            if module:
                os.remove(module.__file__)
```

圖3、圖4、圖5 jit_ops.py 修改段落

1.1 資料預處理(YOLO format to COCO format)

首先如下圖6，先把要讀取的img、json檔案路徑以及要接output json的檔案路徑寫好，接著yolo_to_coco這個function是將原先YOLO format 中groundtruth的bounding box中央點的x y座標比例拉出來進行轉換。換算公式可以理解為x中央點往左或往右半個bounding box寬度即為該bounding box的x_min、x_max，此外根據[3]提到COCO format僅需要[x_min, y_min, width, height]這四個value因此其實可以不用特別去寫x_max、y_max。

```python
import os
import json

yolo_train_dir = './datasets/train_labels'
train_img_dir = './datasets/train2017'
train_output_json = './datasets/annotations/train_coco.json'

yolo_val_dir = './datasets/val_labels'
val_img_dir = './datasets/val2017'
val_output_json = './datasets/annotations/val_coco.json'

def yolo_to_coco(yolo_label, img_width, img_height):
    x_center, y_center, w, h = map(float, yolo_label[1:])
    x_min = (x_center - w / 2) * img_width
    x_max = (x_center + w / 2) * img_width
    y_min = (y_center - h / 2) * img_height
    y_max = (y_center + h / 2) * img_height
    width = w * img_width
    height = h * img_height
    return [x_min, y_min, width, height]#cuz coco only needs xmin and ymin, no needed to show xmax ymax
```

圖6為將YOLO中center原為圖片中比例轉換為真實圖片大小之座標

接著，由於原圖大小為1920x1080大小，然而因為YOLOx-s之輸入圖片大小必為32之倍數，而明顯1080並未能符合其條件。考量其大小比例為16:9，因此放大64倍後將圖片大小調整為1024x576，此舉可以成功保持原圖比例進行訓練，又能夠在未來要生成預測框座標時，僅需要先計算出該座標於1024x576大小圖片之位置後，再放大1.875倍即可。另外轉換為COCO format時需要包含id也就是目標框數量，class_id(0)，bbox則是用來接yolo_to_coco轉出來的[x_min, y_min, width, height]，area則是width, height兩者相乘，而考量除非車子全部擠或疊一起，不然就是把is_crowded設為0。成果圖如下圖7、8，而程式碼部分則是圖9、10


```
100217      {
100218          "id": 6973,
100219          "image_id": 1057,
100220          "category_id": 0,
100221          "bbox": [
100222              854.4,
100223              290.66666666666663,
100224              136.53333333333333,
100225              54.4
100226          ],
100227          "area": 7427.413333333333,
100228          "iscrowd": 0
100229      },
100230      {
100231          "id": 6974,
100232          "image_id": 1057,
100233          "category_id": 0,
100234          "bbox": [
100235              543.4666666666666,
100236              245.33333333333334,
100237              82.13333333333334,
100238              44.266666666666666
100239          ],
100240          "area": 3635.7688888888893,
100241          "iscrowd": 0
100242      },
```

```
100217      {
100218          "id": 6973,
100219          "image_id": 1057,
100220          "category_id": 0,
100221          "bbox": [
100222              854.4,
100223              290.66666666666663,
100224              136.53333333333333,
100225              54.4
100226          ],
100227          "area": 7427.413333333333,
100228          "iscrowd": 0
100229      },
```


圖7、8分別為圖片基本資訊、id為6973、6974皆是源自於1057.jpg(包含各項資訊)

```python
def convert_yolo_to_coco(yolo_dir, output_json, img_dir):
    coco_data = {
        "images": [],
        "annotations": [],
        "categories": [{"id": 0, "name": "car"}],
    }
    ann_id = 1
    img_id = 1
    for img_file in os.listdir(img_dir):
        if img_file.endswith(".jpg"):
            img_path = os.path.join(img_dir, img_file)
            img_width, img_height = 1024, 576
            coco_data["images"].append({
                "id": img_id,
                "file_name": img_file,
                "width": img_width,
                "height": img_height
            })
```

```python
            label_file = os.path.join(yolo_dir, f"{os.path.splitext(img_file)[0]}.txt")
            if os.path.exists(label_file):
                with open(label_file, 'r') as f:
                    for line in f:
                        yolo_label = line.strip().split()
                        bbox = yolo_to_coco(yolo_label, img_width, img_height)
                        coco_data["annotations"].append({
                            "id": ann_id,
                            "image_id": img_id,
                            "category_id": int(yolo_label[0]),  # 類別ID
                            "bbox": bbox,
                            "area": bbox[2] * bbox[3],  # width * height
                            "iscrowd": 0
                        })
                        ann_id += 1
            img_id += 1

    with open(output_json, 'w') as json_file:
        json.dump(coco_data, json_file, indent=4)

convert_yolo_to_coco(yolo_train_dir, train_output_json, train_img_dir)
convert_yolo_to_coco(yolo_val_dir, val_output_json, val_img_dir)

print("Having Generated train_coco.json and val_coco.json")
```

圖9、10為(YOLO format to COCO format)轉換之核心部分

1.2 超參數之設定:

1.2.1 Model Config:

這部分僅更動num_classes為1(因為只有car)，並且設定模型深度、寬度為yolox-s的預設值(0.33、0.50)，至於激活函數依然是設定為SiLU並未做更動。

```
# ---------------- model config ---------------- #
# detect classes number of model
self.num_classes = 1
# factor of model depth
self.depth = 0.33
# factor of model width
self.width = 0.50
# activation name. For example, if using "relu", then "silu" will be replaced to "relu".
self.act = "silu"
```

圖11為model config部分

1.2.2 Dataloader & Transform Config:

此兩部分，分別為引入圖像資料、轉換後(COCO Format)之json file並做Augmentation部分，首先在dataloader僅調整圖片尺寸大小為1024x576(誠如先前提過之保持原圖16:9之比例再進行縮放)並且採用multiscale進行訓練。至於在transform這部分則是遵從source code並未做更動，而在下段也會針對各超參數的使用效果進行說明。

首先，Mosaic、Mixup皆是拼接多張圖片形成一張圖像的方法，其中Mosaic是將多張圖片拼接在一起，至於Mixup則是將兩張圖像以固定比例疊加的方法，此外根據source code，Mosaic每張圖像可以被隨機縮放到10%到200%的大小，Mixup則是每張圖像可以被隨機縮放至50%到150%。hsv_prob則是控制HSV顏色空間變化的機率。至於平移和翻轉則是會在±10%之間進行平移，而翻轉機率設為0.5並且圖像可以在-10度到10度之間隨機旋轉，shear = 2則是可以在±2度範圍間進行隨機裁切。

```
# ---------------- dataloader config ---------------- #
# set worker to 4 for shorter dataloader init time
# If your training process cost many memory, reduce this value.
self.data_num_workers = 4
self.input_size = (1024, 576)  # (height, width)
# Actual multiscale ranges: [640 - 5 * 32, 640 + 5 * 32].
# To disable multiscale training, set the value to 0.
self.multiscale_range = 5
# You can uncomment this line to specify a multiscale range
# self.random_size = (14, 26)
# dir of dataset images, if data_dir is None, this project will use `datasets` dir
self.data_dir = "datasets"
self.train_ann = "train_coco.json"

# name of annotation file for evaluation
self.val_ann = "val_coco.json"
# name of annotation file for testing
#self.test_ann = "instances_test2017.json"
```

```
# --------------- transform config ----------------- #
# prob of applying mosaic aug
self.mosaic_prob = 1.0
# prob of applying mixup aug
self.mixup_prob = 1.0
# prob of applying hsv aug
self.hsv_prob = 1.0
# prob of applying flip aug
self.flip_prob = 0.5
# rotation angle range, for example, if set to 2, the true range is (-2, 2)
self.degrees = 10.0
# translate range, for example, if set to 0.1, the true range is (-0.1, 0.1)
self.translate = 0.1
self.mosaic_scale = (0.1, 2)
# apply mixup aug or not
self.enable_mixup = True
self.mixup_scale = (0.5, 1.5)
# shear angle range, for example, if set to 2, the true range is (-2, 2)
self.shear = 2.0
```

圖12、13為dataloader、transform config部分

1.2.3 Training Config:

考量訓練週期較長，因此將warmup和最後不使用augmentation的epoch比例各佔total_epoch的5%，也就是在訓練的前15epoch為熱身週期(即學習率從0開始逐步往上加)從第16epoch開始從初始的lr並且依據lr_scheduler的調節做調整，直到初始的lr乘上min_lr_ratio即為最終lr。此外，較為特別的是YOLOx的basic_lr_per_img是初始學習率並且會在訓練時乘上batch size，所以參考README後原本64則是改成batch 也就是4，至於testing時圖片尺寸依然保持1024x576，conf則是設定為0.7，nms則是設定為0.65。

```
# --------------  training config --------------------- #
# epoch number used for warmup
self.warmup_epochs = 15
# max training epoch
self.max_epoch = 300
# minimum learning rate during warmup
self.warmup_lr = 0
self.min_lr_ratio = 0.05
# learning rate for one image. During training, lr will multiply batchsize.
self.basic_lr_per_img = 0.01 / 4.0#0.01 / 64.0
# name of LRScheduler
self.scheduler = "yoloxwarmcos"
# last #epoch to close augmention like mosaic
self.no_aug_epochs = 15
# apply EMA during training
self.ema = True
# weight decay of optimizer
self.weight_decay = 5e-4
# momentum of optimizer
self.momentum = 0.9
```

```
self.print_interval = 10
# eval period in epoch, for example,
# if set to 1, model will be evaluate after every epoch.
self.eval_interval = 10
# save history checkpoint or not.
# If set to False, yolox will only save latest and best ckpt.
self.save_history_ckpt = True
# name of experiment
self.exp_name = os.path.split(os.path.realpath(__file__))[1].split(".")[0]

# ----------------  testing config ------------------ #
# output image size during evaluation/test
self.test_size = (1024, 576)
# confidence threshold during evaluation/test,
# boxes whose scores are less than test_conf will be filtered
self.test_conf = 0.70
# nms threshold
self.nmsthre = 0.65
```

圖14、15為training、testing config

1.3 Adding SE Module

參考**sample code**進行撰寫，並且參考一些作法並且加入darknet。此外，有些做法是在backbone中每一層都加入SE Module，也有一種方式是在**最後一層**加入即可。最後選擇如圖17-19於dark5後加入SE Module，加入後如模型結構如圖20所示。

```
class SEModule(nn.Module):
    def __init__(self, in_channels, reduction=16):
        super(SEModule, self).__init__()
        self.in_channels = in_channels
        self.reduction = reduction
        self.avg_pool = nn.AdaptiveAvgPool2d(1)  # 全局平均池化
        self.fc1 = nn.Conv2d(in_channels, in_channels // reduction, kernel_size=1)
        self.relu = nn.ReLU(inplace=True)
        self.fc2 = nn.Conv2d(in_channels // reduction, in_channels, kernel_size=1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        se = self.avg_pool(x)  # 全局平均池化
        se = self.fc1(se)  # 減少通道
        se = self.relu(se)
        se = self.fc2(se)  # 恢復通道
        se = self.sigmoid(se)  # 將權重壓縮到0-1之間
        return x * se  # 將輸入乘以權重
```

```python
class CSPDarknet(nn.Module):
    def __init__(
        self,
        dep_mul,
        wid_mul,
        out_features=("dark3", "dark4", "dark5"),
        depthwise=False,
        act="silu",
        use_se=True,  # 添加SE Module的開關
    ):
        super().__init__()
        assert out_features, "please provide output features of Darknet"
        self.out_features = out_features
        self.use_se = use_se  # 設置SE Module使用與否
```

```python
        # dark5
        self.dark5 = nn.Sequential(
            Conv(base_channels * 8, base_channels * 16, 3, 2, act=act),
            SPPBottleneck(base_channels * 16, base_channels * 16, activation=act),
            CSPLayer(
                base_channels * 16,
                base_channels * 16,
                n=base_depth,
                shortcut=False,
                depthwise=depthwise,
                act=act,
            ),
        )
        if self.use_se:
            self.se5 = SEModule(base_channels * 16)  # 只在dark5層添加SE Module
```

```python
    def forward(self, x):
        outputs = {}
        x = self.stem(x)
        outputs["stem"] = x
        x = self.dark2(x)
        outputs["dark2"] = x
        x = self.dark3(x)
        outputs["dark3"] = x
        x = self.dark4(x)
        #if self.use_se:
        #    x = self.se4(x)  #SE作用於dark4層
        outputs["dark4"] = x
        x = self.dark5(x)
        if self.use_se:
            x = self.se5(x)  # SE作用於dark5層
        outputs["dark5"] = x
        return {k: v for k, v in outputs.items() if k in self.out_features}
```

圖16-19在network blocks中加入SE Module並且在darknet中使用

```
(se5): SEModule(
  (avg_pool): AdaptiveAvgPool2d(output_size=1)
  (fc1): Conv2d(512, 32, kernel_size=(1, 1), stride=(1, 1))
  (relu): ReLU(inplace=True)
  (fc2): Conv2d(32, 512, kernel_size=(1, 1), stride=(1, 1))
  (sigmoid): Sigmoid()
)
)
```

圖20 加入後的 SE Module

2.1 訓練結果(includes with mosaic epochs / without mosaic epochs)

(1-0)根據定義並且添加AP @ IOU=0.85的條件式進入coco_evaluator.py以方便於訓練與驗證時查看效果。

```
306
307             # 計算 AP @ IoU=0.85
308             iou_threshold_index = 7  # IoU=0.85 對應的index
309             precisions = cocoEval.eval['precision']
310             ap_iou_85 = precisions[iou_threshold_index, :, :, 0, -1]  # 取得 IoU=0.85 的 precision
311             ap_iou_85 = np.mean(ap_iou_85[ap_iou_85 > -1])  # 計算有效項的平均 AP
312
313             # 顯示 AP @ IoU=0.85 的結果
314             info += f"Average Precision (AP) @[ IoU=0.85 | area=all | maxDets=100 ] = {ap_iou_85:.3f}\n"
315
```

(1-1)YOLOx-s(Baseline with mosaic epochs）

[(AP) @ IOU = 0.85] = 0.831 (best) @ 203/300 epochs

```
Average forward time: 13.11 ms, Average NMS time: 0.70 ms, Average inference time: 13.82 ms
Average Precision (AP) @[ IoU=0.85 | area=all | maxDets=100 ] = 0.831
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.838
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.987
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.965
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.749
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.855
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.868
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.143
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.864
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.867
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.800
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.882
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.898
per class AP:
| class    | AP      |
|:---------|:--------|
| car      | 83.805 |
per class AR:
| class    | AR      |
|:---------|:--------|
| car      | 86.655 |
```

圖22為YOLOx-s未關閉augment epoch中表現最佳之AP epoch

(1-2)YOLOx-s(Baseline without mosaic epochs）

[(AP) @ IOU = 0.85] = 0.947 (best) @ 297/300 epochs

```
Average forward time: 9.74 ms, Average NMS time: 0.86 ms, Average inference time: 10.60 ms
Average Precision (AP) @[ IoU=0.85 | area=all | maxDets=100 ] = 0.947
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.930
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.990
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.989
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.892
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.939
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.936
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.151
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.940
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.942
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.911
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.950
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.947
per class AP:
| class    | AP      |
|:---------|:--------|
| car      | 93.007 |
per class AR:
| class    | AR      |
|:---------|:--------|
| car      | 94.208 |
Ln 101, Col 9 (41 selected)
```

圖23 為YOLOx-s採用關閉augment epoch中表現最佳之AP epoch

(2-1)YOLOx-s (add SE Module on dark5 with mosaic epochs)

[(AP) @ IOU = 0.85] = 0.828 (best) @ 215/300 epochs

```
Average forward time: 11.11 ms, Average NMS time: 0.66 ms, Average inference time: 11.77 ms
Average Precision (AP) @[ IoU=0.85 | area=all | maxDets=100 ] = 0.828
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.819
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.967
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.946
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.670
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.855
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.864
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.142
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.846
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.847
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.714
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.879
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.889
per class AP:
| class   | AP     |
|:--------|:-------|
| car     | 81.902 |
per class AR:
| class   | AR     |
|:--------|:-------|
| car     | 84.659 |
```

圖24 為加入SE Module 未關閉augment epoch中表現最佳之AP epoch

(2-2)YOLOx-s (add SE Module on dark5 without mosaic epochs)

[(AP) @ IOU = 0.85] = 0.947 (best) @ 300/300 epochs

```
2024-10-18 20:32:09 | INFO     | yolox.layers.fast_coco_eval_api:266 - DONE (t=0.02s).
2024-10-18 20:32:09 | INFO     | yolox.core.trainer:416 -
Average forward time: 10.49 ms, Average NMS time: 0.55 ms, Average inference time: 11.04 ms
Average Precision (AP) @[ IoU=0.85 | area=all | maxDets=100 ] = 0.947
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.935
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.989
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.989
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.911
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.942
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.937
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.151
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.943
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.945
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.924
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.950
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.949
per class AP:
| class   | AP     |
|:--------|:-------|
| car     | 93.500 |
per class AR:
| class   | AR     |
|:--------|:-------|
| car     | 94.471 |

2024-10-18 20:32:09 | INFO     | yolox.core.trainer:437 - Save weights to ./YOLOX_outputs\yolox_s
2024-10-18 20:32:09 | INFO     | yolox.core.trainer:437 - Save weights to ./YOLOX_outputs\yolox_s
2024-10-18 20:32:09 | INFO     | yolox.core.trainer:200 - Training of experiment is done and the best AP is 93.50
```

圖25 為加入SE Module 採用關閉augment epoch中表現最佳之AP epoch

2.2 成果展示

以下兩張圖為使用測試集中之成果照片，可看出圖26右方其實有輛汽車，不過觀察其類似場景之ground truth並未進行標記。不過本資料集中亦有部分是因為遮擋而在ground truth就並未標記，因此有少部分會被model判斷為存在的情形。

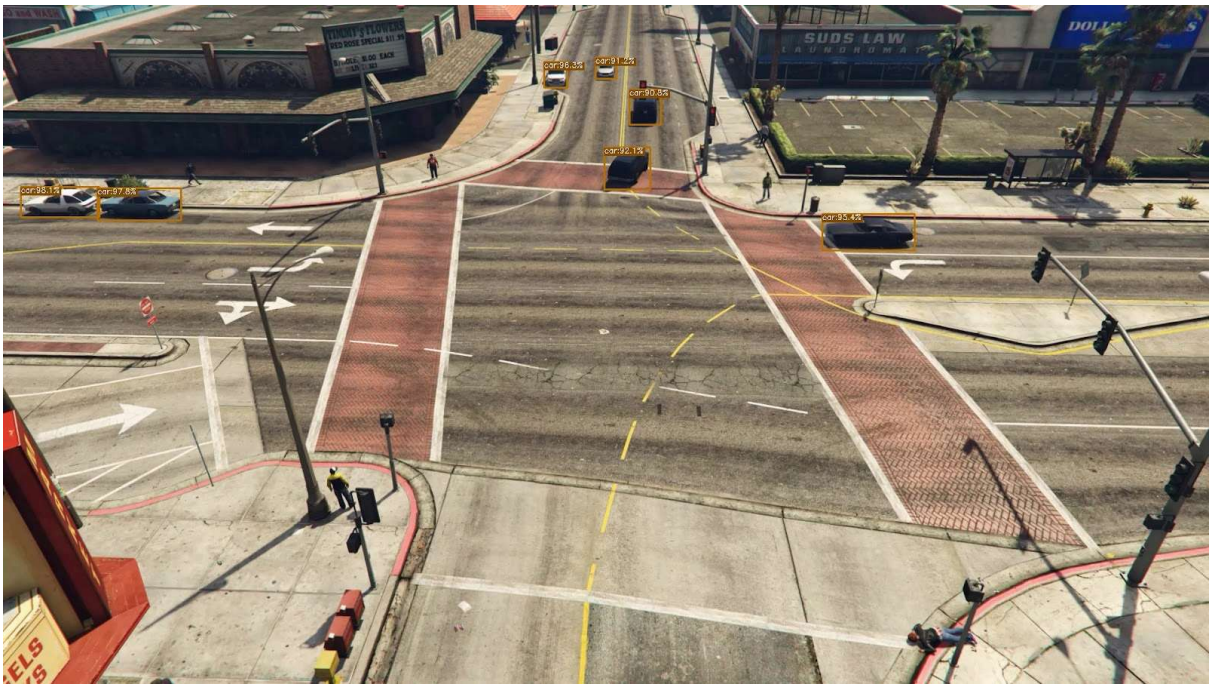


圖26、27測試集中經過demo後畫面

2.3 將生成之結果轉換為txt檔案

這邊需要特別留意的是因為可能因為檔案解析度較大，在轉換456張時容易cuda out memory所以必要時需要切分test分為兩份(如228、228張)再進行轉換。此外先設定conf、nms的閾值為0.7、0.65，並且由於模型當初是輸入1024x576因此要轉換即乘上1.875倍，再分成456個txt檔案並填入。

```python
import os
import torch
import cv2
from pathlib import Path
from yolox.exp import get_exp
from yolox.data.data_augment import ValTransform
from yolox.utils import postprocess

result_dir = Path('Result')
result_dir.mkdir(parents=True, exist_ok=True)

exp = get_exp('exps/example/custom/yolox_s.py', 'yolox-s')
model = exp.get_model()
ckpt = torch.load('YOLOX_outputs/yolox_s/best.pth', map_location='cpu')
model.eval()
model.cuda()
preproc = ValTransform(legacy=False)

image_dir = Path('datasets/test')
image_paths = list(image_dir.glob('*.jpg'))

# 讀取和預處理圖像
imgs = []
original_sizes = []
for path in image_paths:
    img = cv2.imread(str(path))
    original_sizes.append(img.shape[:2])  # 儲存原始圖像尺寸 (height, width)
    img, _ = preproc(img, None, exp.test_size)  # 這裡 exp.test_size 是 1024x576
    imgs.append(torch.from_numpy(img).unsqueeze(0))
batch = torch.cat(imgs, 0).cuda()
```

```python
# inference
with torch.no_grad():
    outputs = model(batch)
    # threshold: conf = 0.7 nms = 0.65
    outputs = postprocess(outputs, exp.num_classes, 0.7, 0.65)

# 為每張圖片生成獨立的txt文件，並保存結果
for i, output in enumerate(outputs):
    image_name = image_paths[i].stem
    txt_file_path = result_dir / f"{image_name}.txt"

    with open(txt_file_path, "w") as f:
        if output is not None:
            output = output.cpu()
            bboxes = output[:, 0:4]
            scores = output[:, 4] * output[:, 5]
            original_height, original_width = original_sizes[i]
            scale_w = original_width / exp.test_size[1]  # 寬度縮放比例
            scale_h = original_height / exp.test_size[0]  # 高度縮放比例
            for j in range(bboxes.size(0)):
                bbox = bboxes[j]
                score = scores[j]
                # 邊界框從模型輸入比例（1024x576）轉換為原始圖像比例
                bbox[0] *= scale_w  # x1
                bbox[1] *= scale_h  # y1
                bbox[2] *= scale_w  # x2
                bbox[3] *= scale_h  # y2
                f.write(f"0 {score:.6f} {bbox[0]:.2f} {bbox[1]:.2f} {bbox[2]:.2f} {bbox[3]:.2f}\n")

            print(f"Complete and save {txt_file_path}")
```

圖28、29 將結果圖轉成result.txt

2.4 Command Line

0. yolo to coco:

python yolo_to_coco.py

1. Training:

python tools/train.py -f exps/example/custom/yolox_s.py -d 0 -b 4 --fp16 -o

2. Demo:

python tools/demo.py image -n yolox-s --path "datasets/test" --ckpt "313553024_ckpt.pth" --conf 0.7 --nms 0.65 --device gpu --save_result --save_result

3. 將輸出轉為txt:

python to_txt.py

2.5 Other Implementations

參考[4]使用CBAM 注意力機制將dark3~dark5皆接上了CBAM注意力機制，不同的是，我在dark5後面仍保持使用SE Module而dark3、dark4則是加入CBAM。

```python
class ChannelAttention(nn.Module):
    def __init__(self, in_planes, ratio=8):
        super(ChannelAttention, self).__init__()
        self.avg_pool = nn.AdaptiveAvgPool2d(1)
        self.max_pool = nn.AdaptiveMaxPool2d(1)

        # 利用1x1卷积代替全连接
        self.fc1 = nn.Conv2d(in_planes, in_planes // ratio, 1, bias=False)
        self.relu1 = nn.ReLU()
        self.fc2 = nn.Conv2d(in_planes // ratio, in_planes, 1, bias=False)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        avg_out = self.fc2(self.relu1(self.fc1(self.avg_pool(x))))
        max_out = self.fc2(self.relu1(self.fc1(self.max_pool(x))))
        out = avg_out + max_out
        return self.sigmoid(out)
```

```python
class SpatialAttention(nn.Module):
    def __init__(self, kernel_size=7):
        super(SpatialAttention, self).__init__()

        assert kernel_size in (3, 7), 'kernel size must be 3 or 7'
        padding = 3 if kernel_size == 7 else 1
        self.conv1 = nn.Conv2d(2, 1, kernel_size, padding=padding, bias=False)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        avg_out = torch.mean(x, dim=1, keepdim=True)
        max_out, _ = torch.max(x, dim=1, keepdim=True)
        x = torch.cat([avg_out, max_out], dim=1)
        x = self.conv1(x)
        return self.sigmoid(x)
```

```python
# CBAM attn
class CBAModule(nn.Module):
    def __init__(self, channel, ratio=8, kernel_size=7):
        super(CBAModule, self).__init__()
        self.channelattention = ChannelAttention(channel, ratio=ratio)
        self.spatialattention = SpatialAttention(kernel_size=kernel_size)

    def forward(self, x):
        out = self.channelattention(x) * x
        out = self.spatialattention(out) * out
        return out
```

圖30~32為新加入之CBAM

經過300Epochs訓練並且保持相同超參數後，得到之以下結果。

YOLOx-s (add CBAM on dark3、dark4 and add SE Module on dark5 with mosaic epochs)

[(AP) @ IOU = 0.85] = 0.854 (best) @ 113/300 epochs

```
2024-10-19 12:35:53.354 | INFO     | yolox.core.trainer:evaluate_and_save_model:416 -
Average forward time: 9.89 ms, Average NMS time: 0.46 ms, Average inference time: 10.35 ms
Average Precision (AP) @[ IoU=0.85 | area=all | maxDets=100 ] = 0.854
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.846
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.978
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.966
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.737
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.870
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.878
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.145
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.871
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.871
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.771
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.895
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.906
per class AP:
| class   | AP     |
|:--------|:-------|
| car     | 84.570 |
per class AR:
| class   | AR     |
|:--------|:-------|
| car     | 87.087 |
```

圖33 為加入CBAM、SE Module 未關閉augment epoch中表現最佳之AP epoch

YOLOx-s (add CBAM on dark3、dark4 and add SE Module on dark5 without mosaic epochs)

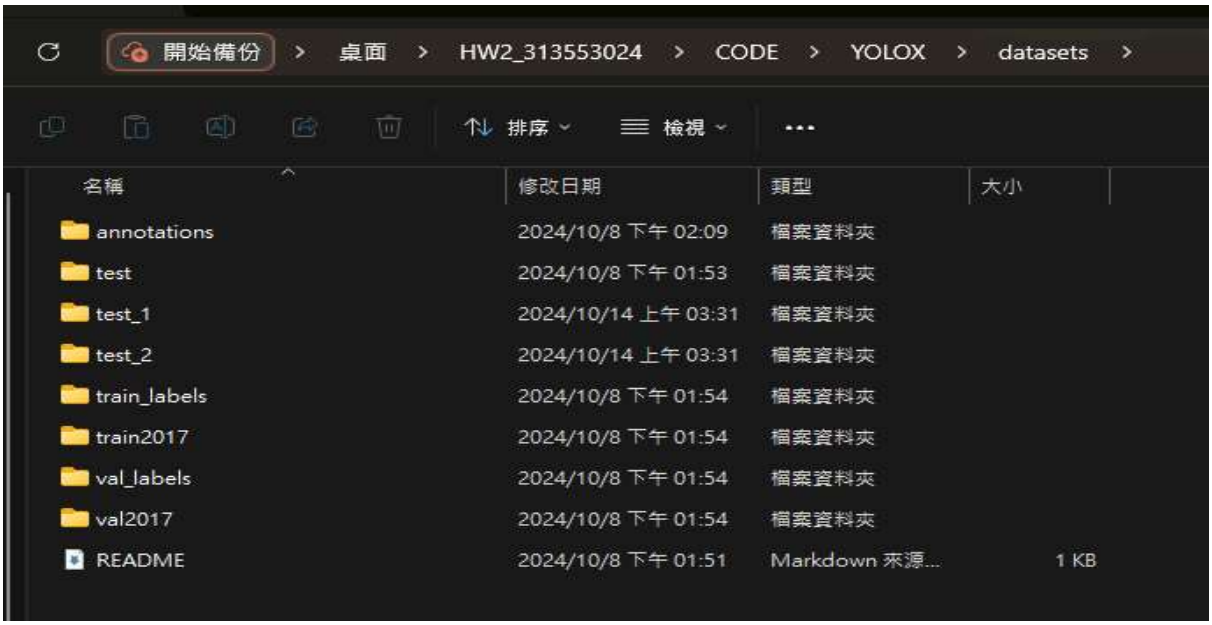
[(AP) @ IOU = 0.85] = 0.946 (best) @ 300/300 epochs

```
Average forward time: 9.84 ms, Average NMS time: 0.66 ms, Average inference time: 10.50 ms
Average Precision (AP) @[ IoU=0.85 | area=all | maxDets=100 ] = 0.946
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.933
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.990
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.989
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.910
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.937
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.939
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.152
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.939
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.923
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.946
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.950
per class AP:
| class   | AP     |
|:--------|:-------|
| car     | 93.304 |
per class AR:
| class   | AR     |
|:--------|:-------|
| car     | 94.181 |
```

圖34 為加入CBAM、SE Module 採用關閉augment epoch中表現最佳之AP epoch

補充說明:

本次作業資料集檔案結構如下圖35、36，annotations用來存放轉換後的train、val_label json檔案，而test_1、test_2則是因為測試集456張，因為輸入解析度較大因此在執行to_txt.py時會記憶體不足所切割，並未新增或刪改。至於原先作業提供之資料集為train2017、val2017、test。



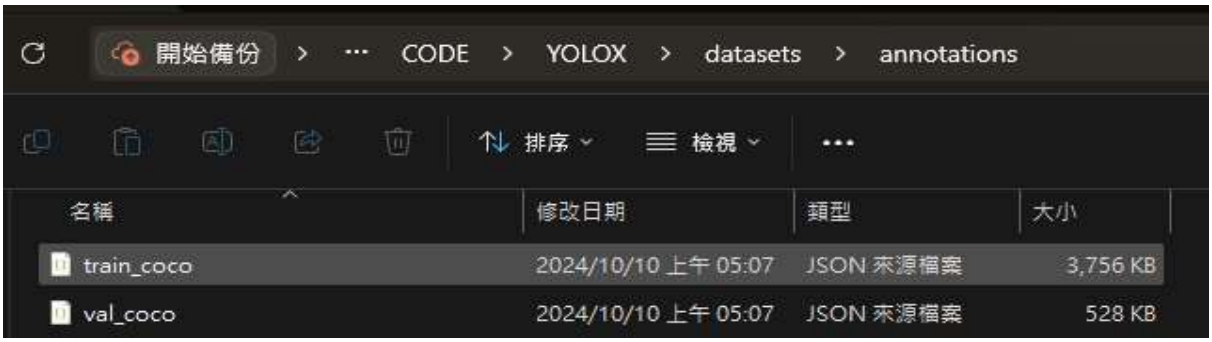


圖35、36 為資料集存放之檔案結構(因應要求並未放置在E3繳交區)

先前第一段提到之問題還有找不到C++編譯器，具體解法則是下載pybind11 並且在VSCODE底下創建c_cpp_properties.json中加入pybind11之路徑如8、9行

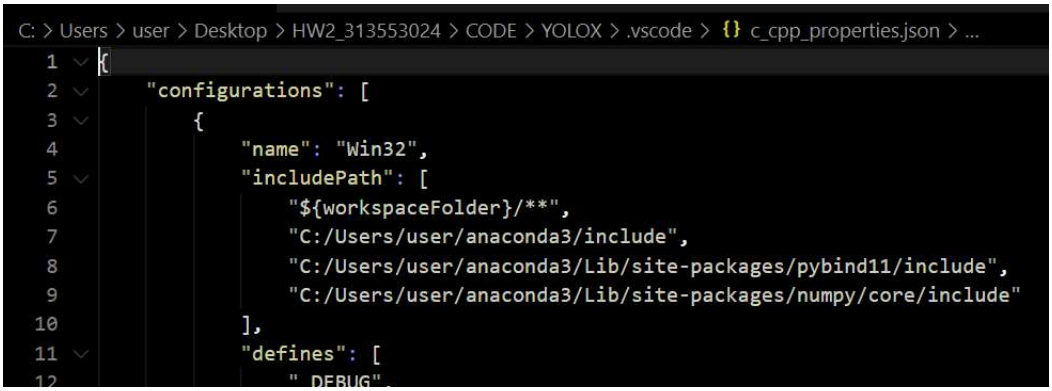


圖37 c_cpp_properties.json

Reference

[1]張家銘. (2021, September 4). *如何使用自己的資料集訓練 YOLOX*. Medium. https://d246810g2000.medium.com/%E5%A6%82%E4%BD%95%E4%BD%BF%E7%94%A8%E8%87%AA%E5%B7%B1%E7%9A%84%E8%B3%87%E6%96%99%E9%9B%86%E8%A8%93%E7%B7%B4-yolox-c02548734a48

[2]https://hackmd.io/@yeeen260/yolox?utm_source=preview-mode&utm_medium=rec

[3]https://developer.aliyun.com/article/1078304

[4]https://blog.csdn.net/m0_70388905/article/details/126002838