

# Unified-OneHead Multi-Task Challenge

Q36134255 電通所 碩一 郭人瑋

## 1. 簡介 (Introduction)

本專案旨在應對「Unified-OneHead Multi-Task Challenge」，設計並實作一個單一的深度學習模型，使其能夠同時處理語意分割 (Semantic Segmentation)、物件偵測 (Object Detection) 和圖片分類 (Image Classification) 三項不同的視覺任務。

專案的核心挑戰在於，模型需要透過循序訓練 (Sequential Training) 的方式依次學習這三項任務，同時必須克服在學習新任務時遺忘舊任務知識的「災難性遺忘 (Catastrophic Forgetting)」問題。根據作業要求，最終所有任務的性能指標相對於其基準線，下降幅度不得超過 5%。

## 2. 架構設計 (Model Architecture)

為了在滿足多任務需求的同時，兼顧資源效率（參數 < 8M，推論 < 150ms），

任務要求：

Key Point	Requirement
Task	Build <b>one head (2-3 layers)</b> that outputs detection + segmentation + classification simultaneously
Data	Three <i>mutually exclusive</i> mini-datasets (< 120 MB total) supplied by the course; one-command download
Hardware	Free Google Colab GPU (T4 or V100) • <b>Total training time ≤ 2 h</b>
Core difficulty	<i>Catastrophic forgetting</i> : after alternating training, each task must drop ≤ 5 % vs. its single-task baseline

編譯環境：Google colab

硬體：GPU：T4

## 變更執行階段類型

執行階段類型

Python 3

硬體加速器

- ☐ CPU
- ☐ A100 GPU
- ☐ L4 GPU
- ☒ T4 GPU
- ☐ v6e-1 TPU
- ☐ v5e-1 TPU
- ☐ v2-8 TPU

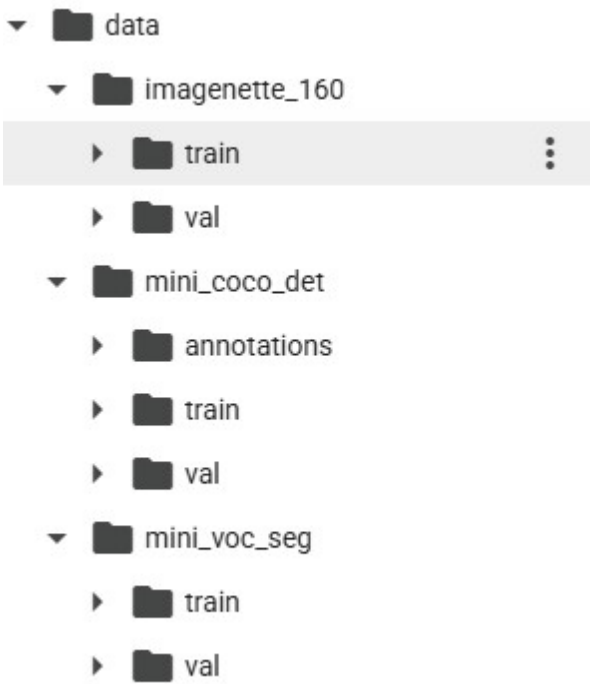
大量 RAM

取消 儲存

資料集 Datasets :

Task	Subset Name	Origin	Download Size	Images (train/val)	Annotation
Detection	<b>Mini-COCO-Det</b>	COCO 2017 (10 classes)	45 MB	300 (240 / 60)	COCO JSON
Segmentation	<b>Mini-VOC-Seg</b>	PASCAL VOC 2012	30 MB	300 (240 / 60)	PNG masks
Classification	<b>Imagenette-160</b>	Imagenette v2	25 MB	300 (240 / 60)	folder / label

實際資料集結構 :



Step(1) : 清理與建立目錄結構

建立一套全新的、有組織的資料夾結構，分別用來存放三個任務（分類、分割、偵測）的訓練和驗證資料，以及一個暫存區。

Step(2) : 下載與解壓縮

程式內建了三個資料集（Imagenette v2, PASCAL VOC 2012, COCO2017）的下載網址，下載完畢後，自動解壓縮這些壓縮檔（.zip, .tar, .tgz）到暫存區，方便後續處理。

Step(3) : 隨機抽樣處理三個不同的資料集

分別處理三個不同的資料集，從中各抽出 **240 張** 作為訓練集 (train)，**60 張** 作為驗證集 (val)。

- **Imagenette (影像分類任務)**：建立一個小型的圖片分類資料集。

**作法**：從解壓縮後的大量圖片中，隨機選取 300 張。然後將其中 240 張複製到 `imagenette_160/train`，剩下的 60 張複製到 `imagenette_160/val`。複製時會**保留它原本的分類資料夾結構**，這樣模型才知道哪張圖屬於哪個類別。

- **PASCAL VOC (語意分割任務)**：建立一個小型的圖像分割資料集，每張圖都有對應的 mask。

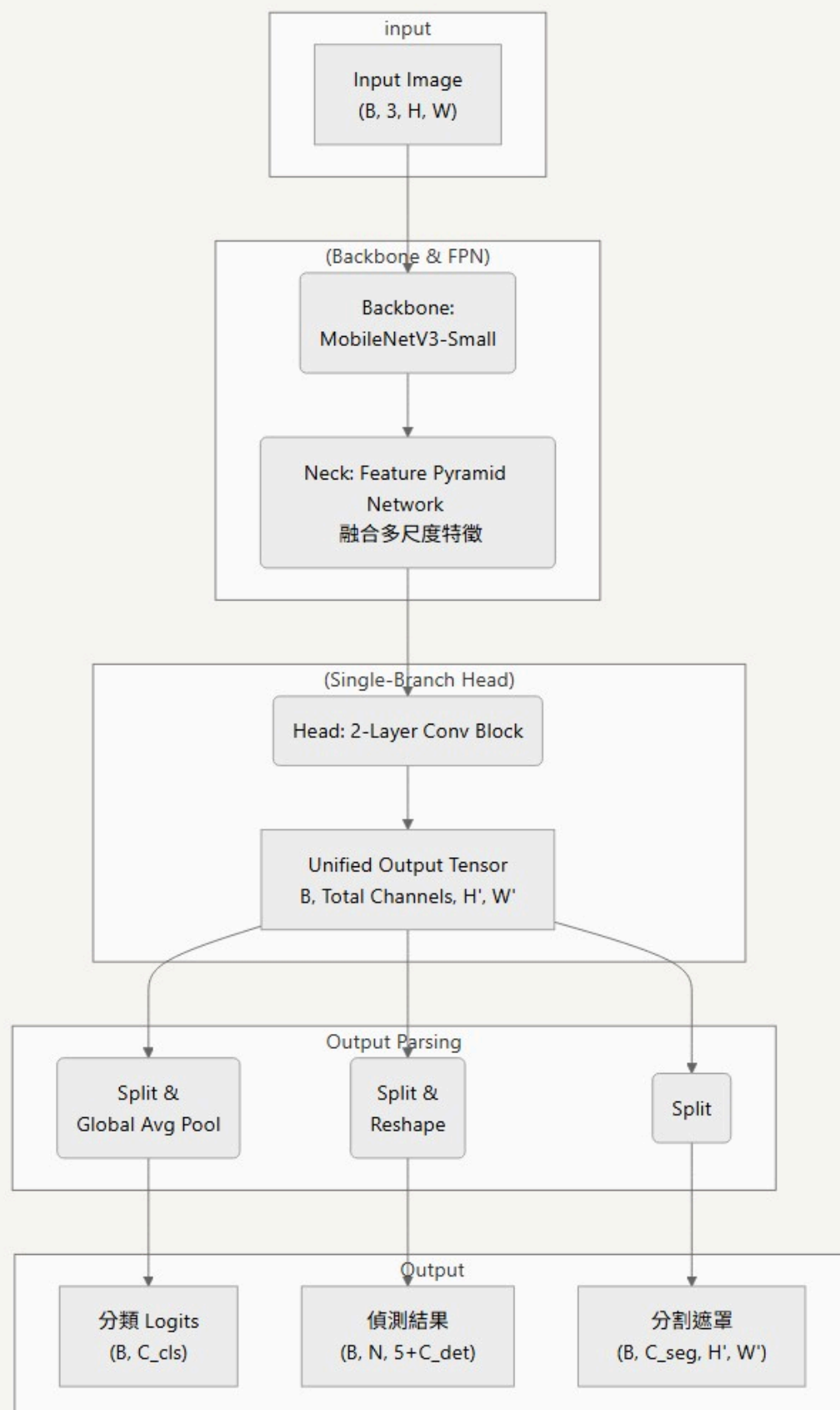
**作法**：確保只選取那些**同時擁有原始圖和答案圖**的圖片，並且從中隨機選取 300 張，一樣是 240 張及對應的答案圖複製到 `mini_voc_seg/train`，60 張複製到 `mini_voc_seg/val`。

- **COCO (物件偵測任務)**：建立一個小型的物件偵測資料集，包含圖片和標註檔 (Annotation)。

**作法**：隨機選取 300 張圖片，分別複製到 `mini_coco_det` 的 train 和 val 資料夾，接著讀取原始的、**包含數萬筆紀錄的 COCO 標註檔 (.json)**，然後**只挑出與這 300 張抽樣圖片相關的標註訊息**，為訓練集和驗證集分別產生**專屬的、輕量化的新標註檔**，減少了後續讀取資料的負擔。

Step(4)：清理過程中暫存檔案，保持空間整潔

接下來我設計了如下的統一模型架構



備註：

- 偵測結果 (B, N, 5+C<sub>det</sub>)：N 指的是預測出的偵測框總數，5：代表位置和大小 (cx, cy, w, h) 以及信心度 (conf)，表示這個輸出張量的形狀是 [批次大小, 偵測框數量, 每個框的資訊維度]。
- 分類 Logits (B, C<sub>cls</sub>)：每一張輸入圖片（在一個批次 B 中）輸出一組長度為 C<sub>cls</sub> 的預測分數，表示這個輸出張量的形狀是 [批次大小, 類別數量]。
- 分割遮罩 (B, C<sub>seg</sub>, H', W')：C<sub>seg</sub> 是分割的總類別數，H' × W' 是輸出遮罩的高度和寬度，表示這個輸出張量的形狀是 [批次大小, 分割類別數, 高度, 寬度]。

程式架構：

```
class UnifiedModel(nn.Module):
    def __init__(self, num_classes_cls=10, num_classes_det=10, num_classes_seg=21):
        super(UnifiedModel, self).__init__()
        # 1. Backbone
        backbone = models.mobilenet_v3_small(weights=models.MobileNet_V3_Small_Weights.IMAGENET1K_V1)
```



```

self.backbone = backbone.features
# 2. Neck
self.fpn = FeaturePyramidNetwork(
    in_channels_list=[24, 40, 576],
    out_channels=128,
    extra_blocks=LastLevelMaxPool()
)
# --- 3. Head 設計 ---
# 計算統一輸出層需要多少個 channel
# 偵測: 每個像素點預測 (cx, cy, w, h, conf) 5個值 + C_det 個類別分數
# 根據作業要求, 偵測輸出為 N x (cx, cy, w, h, conf, C_det)
# 這裡我們假設 C_det 就是 num_classes_det
det_channels = 5 + num_classes_det
# 分割: C_seg 個類別
seg_channels = num_classes_seg
# 分類: C_cls 個類別
cls_channels = num_classes_cls

total_output_channels = det_channels + seg_channels + cls_channels

# "Exactly 2-3 layers" + "single branch"
self.single_branch_head = nn.Sequential(
    # 第 1 層: 特徵提煉 (Conv + ReLU)
    nn.Conv2d(in_channels=128, out_channels=128, kernel_size=3, padding=1),
    nn.ReLU(),
    # 第 2 層: 統一輸出層 (Conv)
    nn.Conv2d(in_channels=128, out_channels=total_output_channels, kernel_size=1)
)
# 儲存 channel 數量, 方便 forward 中使用
self.det_channels = det_channels
self.seg_channels = seg_channels
self.cls_channels = cls_channels
self.classifier = nn.Linear(1, num_classes_cls)
def forward(self, x):
    # Backbone 提取特徵
    feats = {}
    for i, layer in enumerate(self.backbone):
        x = layer(x)
        # 根據 FPN 的要求提取特定層的輸出
        if i in [3, 6, 12]: feats[f'feat{i}'] = x
    # Neck (FPN) 融合特徵
    fpn_feats = self.fpn(feats)
    fpn_out = fpn_feats['feat3'] # 使用 FPN 的輸出, Shape: [B, 128, H/4, W/4]
    # Head (Single Branch) 產生統一輸出
    unified_output = self.single_branch_head(fpn_out) # Shape: [B, total_channels, H/4, W/4]
    # --- 對統一輸出進行切割與解析 ---
    # 按照 channel 維度進行切割
    split_sizes = [self.seg_channels, self.det_channels, self.cls_channels]
    seg_output_map, det_output_map, cls_output_map = torch.split(unified_output, split_sizes, dim=1)

    # 1. 處理分類輸出
    # 對分類特徵圖進行全域平均池化, 得到分類 logits
    cls_logits = torch.mean(cls_output_map, dim=[2, 3])

    # 2. 處理分割輸出
    # 分割圖可以直接使用
    seg_mask = seg_output_map

```

```
# 3. 處理偵測輸出
# 調整形狀以符合 N x (cx, cy, w, h, conf, C_det) 的格式
B, _, H, W = det_output_map.shape
det_output = det_output_map.permute(0, 2, 3, 1).contiguous().view(B, -1, self.det_channels)

# 按照規定，一次返回所有輸出
return cls_logits, det_output, seg_mask
```

2.1. 架構概述

為了滿足多任務學習的需求並符合所有設計規範，模型採用了共享特徵提取層和統一輸出頭的設計。其架構主要由以下三個核心部份組成：

- **共享主幹 (Shared Backbone):** 負責從輸入圖片中提取底層特徵。
- **共享頸部 (Shared Neck):** 透過特徵金字塔網路 (FPN) 融合來自骨幹網路不同層級的特徵，以生成富含多尺度資訊的特徵圖。
- **單一支頭 (Single-Branch Head):** 接收由 Neck 產生的特徵圖，通過一個包含 2 層卷積的**單一線性分支**，一次性地產生一個包含所有三個任務（分類、偵測、分割）預測資訊的**統一輸出張量 (Unified Output Tensor)**。

2.2. 各模組詳述

1. 主幹網路 (Backbone Network)

- **基礎模型:** 選用輕量級且高效的 **MobileNetV3-Small** 作為特徵提取的主幹。
- **預訓練權重:** 載入在 ImageNet 資料集上預訓練的權重，這為模型提供了一個強大的初始特徵提取能力。

2. 頸部：特徵金字塔網路 (Neck: Feature Pyramid Network)

- **技術:** 採用**特徵金字塔網路 (FPN)** 作為模型的 Neck，符合了 **a single FPN layer** 的規定。
- **功能:** FPN 從主幹網路的不同深度（第 3、6、12 層）提取特徵圖，並將這些高層語意特徵與低層細節特徵進行融合。這使得模型能夠產生對不同尺度物件都具有良好表達能力的特徵圖，為後續的統一頭部提供高品質的輸入。

3. 單一支頭 (Single-Branch Head)

為了滿足 **Exactly 2-3 layers and single branch** 的規定，我設計了一個全新的統一輸出頭，取代了傳統的多頭設計。

- **結構:**
  1. 此 Head 由一個包含**兩層卷積網路**的 **nn.Sequential** 模組構成，完全符合層數和單一支的要求。
  2. 第一層卷積用於對 FPN 輸出的特徵進行提煉。
  3. **第二層（也是最後一層）卷積是核心**，它一次性地輸出一張巨大的特徵圖，其**頻道數 (Total Channels)** 是三個任務所需頻道數的總和：

$$TotalChannels = Channels_{seg} + Channels_{det} + Channels_{cls}$$

- **輸出解析 (Output Parsing):**

神經網路的計算在產生上述的「統一輸出張量」後即告完成。接下來，在 **forward** 方法中，我們透過程式碼邏輯對其進行解析：

  1. **切割:** 使用 **torch.split** 沿著頻道維度將統一張量切割成三份，分別對應分割、偵測和分類的原始預測圖。
  2. **塑形:** 對各個任務的預測圖進行必要的後處理（如對分類圖進行全域平均池化，對偵測圖進行維度重排與變形），使其最終格式完全規定的 Output。

經過測試，本模型的資源使用情況完全在作業要求範圍內：

( 20 Epoch )

- ✓ 模型參數數量：1,605,602 個 (< 8M)
- ✓ 單張推論時間：6.67 ms (≤ 150 ms)
- ✓ 總訓練時間：256.47 秒 (≤ 2 小時)

=====

( 100 Epoch )

- ✓ 模型參數數量：1,605,602 個 (< 8M)
- ✓ 單張推論時間：8.14 ms ( $\leq$  150 ms)
- ✓ 總訓練時間：1293.63 秒 ( $\leq$  2 小時)

評估項目	作業要求	程式成果
總參數	< 8 M	~1.6 M
推論速度	$\leq$ 150 ms	~6.67 or 8.14 ms

### 3. 訓練策略與流程 (Training Strategy and Flow)

為了讓單一模型能循序地掌握三種不同的視覺任務，並有效應對災難性遺忘 (Catastrophic Forgetting) 的挑戰，我們設計了一套結合循序分階段訓練 (Sequential Staged Training) 與經驗回放 (Experience Replay) Replay buffer 的複合式訓練策略。

訓練階段要求：

```
Stage 0 (optional) - warm-up / ImageNet pretrain
Stage 1 - Train ONLY on Dataset A (Seg) → record mIoU_base
Stage 2 - Train ONLY on Dataset B (Det) → measure mIoU_drop
Stage 3 - Train ONLY on Dataset C (Cls) → measure mIoU_drop + mAP_drop
```

#### 3.1. 訓練排程

本專案嚴格遵守作業規定的三階段循序訓練排程：

1. **Stage 1:** 只在 Mini-VOC-Seg 資料集上訓練，目標是讓模型學會語意分割，並記錄 `mIoU_base`。
2. **Stage 2:** 接著只在 Mini-COCO-Det 資料集上訓練，讓模型學習物件偵測。
3. **Stage 3:** 最後只在 Imagenette-160 資料集上訓練，讓模型學習圖片分類。

#### 3.2. 第一階段：專注語意分割 (Baseline Training)

- **目標：**讓模型從零開始，專注學習第一項任務：語意分割。
- **資料集：**`VOCSegmentationDataset`。
- **損失函數：**`nn.CrossEntropyLoss`，計算模型輸出的分割遮罩與真實遮罩之間的差異。
- **流程與目的:**在此階段，整個模型的所有參數都會針對分割任務進行優化。主要為了教會模型如何進行像素級別的分類，同時為模型建立**基準 (Baseline)**；訓練完成後，會記錄下此時模型在驗證集上達到的最佳 `mIoU` 分數，作為後續階段評估遺忘程度的**基準性能 ( `mIoU_base` )**。

#### 3.3. 第二階段：學習物件偵測與經驗回放

- **目標：**在不嚴重損害分割性能的前提下，教會模型新的技能：物件偵測。
- **資料集：**主要任務: `MiniCocoDetection` (偵測資料)；回放任務: `VOCSegmentationDataset` (分割資料)。
- **複合式損失函數：**在此階段，每一個訓練批次 (batch) 都由偵測資料和一小部分回放的分割資料組成。總損失是兩個部分損失的加權總和： $L_{total} = L_{detection} + \lambda_{replay} \cdot L_{segmentation_{replay}}$
- $L_{detection}$ ：使用 `nn.MSELoss` 計算模型預測的邊界框與真實邊界框之間的定位誤差。
- $L_{segmentation_{replay}}$ ：使用 `nn.CrossEntropyLoss` 計算模型在回放的分割資料上的表現。
- $\lambda_{replay}$ ：程式中的 `replay_loss_weight` 超參數，用來平衡學習新知識與鞏固舊知識的重要性。
- **流程與目的：**模型的主力在於學習偵測任務，但經驗回放的損失項  $L_{segmentation_{replay}}$ ，不停地將模型的特徵提取層拉回到能同時理解分割任務的狀態，從而有效減緩對第一階段知識的遺忘。

#### 3.3. 第三階段：學習影像分類與經驗回放

- **目標：**加入第三項也是最後一項技能：影像分類，並持續對抗遺忘。
- **資料集：**主要任務: `ImageFolder` (分類資料)；回放任務: `VOCSegmentationDataset` (分割資料)。
- **複合式損失函數：**

- 總損失的計算方式與第二階段類似：

$$L_{total} = L_{classification} + \lambda_{replay} \cdot L_{segmentation_{replay}}$$

- $L_{classification}$ ：用 `nn.CrossEntropyLoss` 計算模型對整張圖片的分類預測與真實標籤的誤差。
- $L_{segmentation_{replay}}$ ：與第二階段完全相同，持續鞏固分割能力。
- **流程與目的**：在模型學習最高層級的語意概念（影像分類）時，經驗回放機制依然在運作，確保底層的像素級和物件級特徵識別能力不會因為學習新任務而崩潰。

### 3.4. 災難性遺忘問題與解方

循序學習的核心挑戰是災難性遺忘。為了驗證並解決此問題，我進行了兩組對照實驗。

- **對照組 (無經驗回放)**: 直接進行三階段訓練。預期會觀察到分割任務的性能在 Stage 2 和 3 大幅下降。
- **實驗組 (有經驗回放)**: 採用「經驗回放 (Replay Buffer)」作為抗遺忘策略。在 Stage 2 和 3 訓練新任務時，每個批次中會混入少量（4張）Stage 1 的分割資料，讓模型在學習新知的同時「複習」舊技能。混合 Loss 的計算方式為 `Total_Loss = Loss_New_Task + 0.8 * Loss_Replay_Task`。

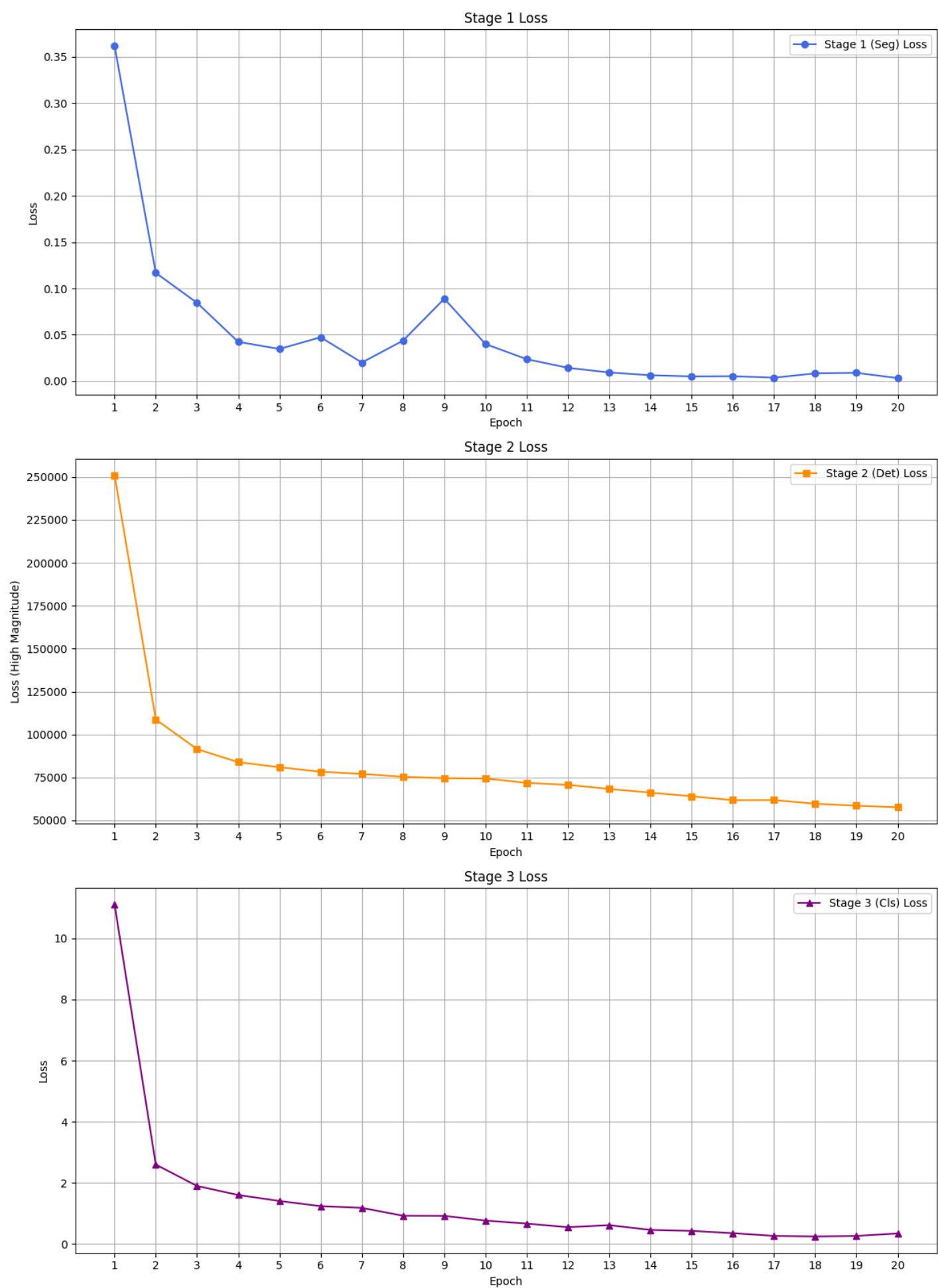
## 4. 實驗結果與分析(Experimental results and analysis)

### 4.1. 訓練損失分析 (Training Loss Analysis)

(Epochs=20)



Training Loss Across All Stages (With Experience Replay)



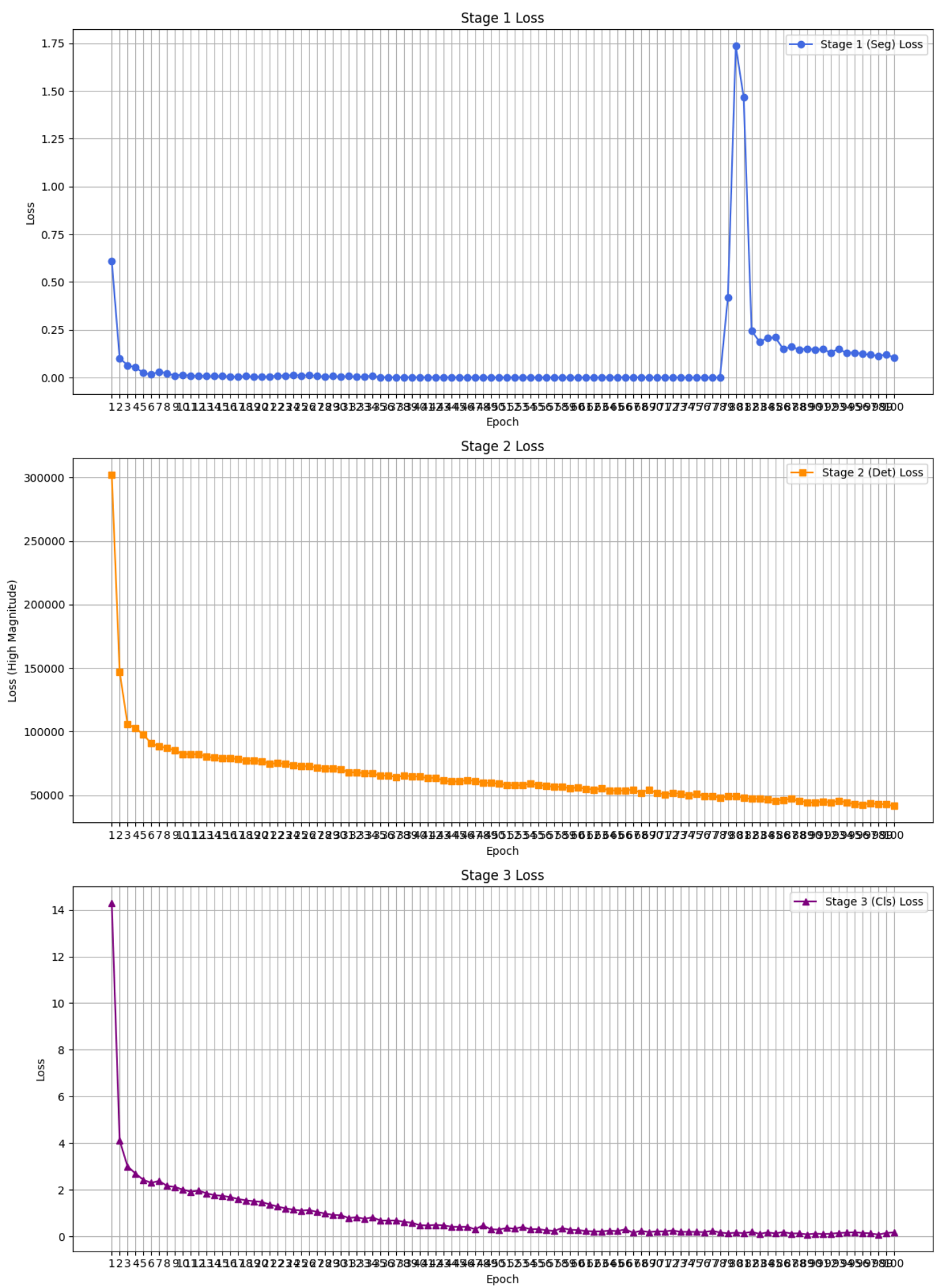
上圖展示了模型在三個訓練階段中，各自主要任務的損失函數變化情況。

- **第一階段 (分割任務):** 藍色的曲線顯示，分割損失 (Seg Loss) 在前 4 個 Epoch 內迅速下降，並在後續訓練中穩定收斂到一個非常低的水平（約 0.01-0.05）。這表明模型在此階段有效地學習並掌握了語意分割任務。
- **第二階段 (偵測任務):** 橙色的曲線代表偵測損失 (Det Loss)。其數值（Y軸）遠高於其他兩個階段，這是因為偵測任務中的邊界框回歸通常使用均方誤差 (MSE) 類的損失函數。儘管數值高，但曲線呈現出清晰且穩定的下降趨勢，證明模型正在成功學習物件偵測。
- **第三階段 (分類任務):** 紫色的曲線顯示，分類損失 (Cls Loss) 同樣在前幾個 Epoch 內急劇下降，並快速收斂。這說明模型能夠迅速地掌握最高層級的影像分類任務。

結果與討論：三個階段的損失曲線均成功收斂，證明我們的訓練流程是有效的，模型在每個階段都能學到對應的主要任務。

(Epochs=100)

Training Loss Across All Stages (With Experience Replay)



從三階段的損失曲線來看，模型在各個階段都表現出學習的趨勢：

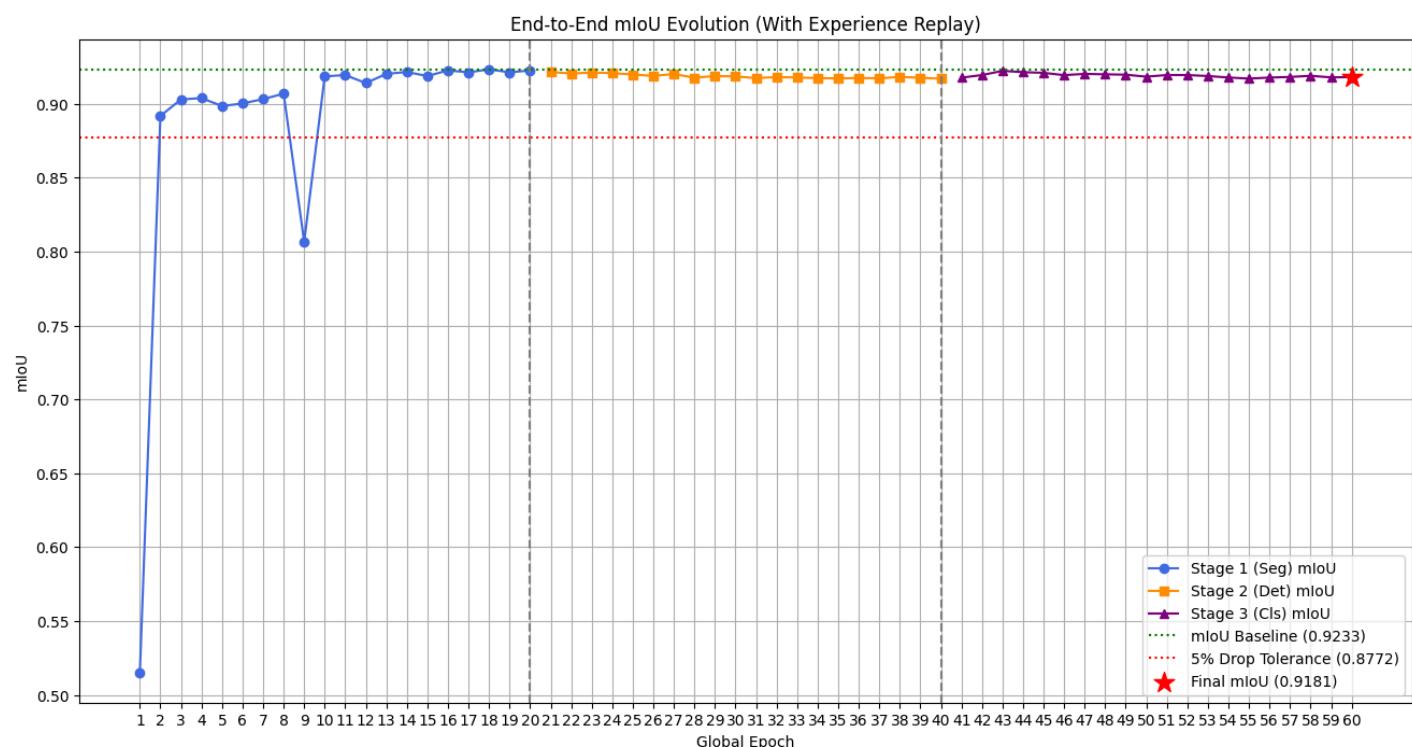
- **第一階段 (分割任務):** 損失在前幾個 Epoch 迅速下降，證明了快速的初步學習。然而，值得注意的是，在訓練後期（約 Epoch 75-80），損失曲線出現了一次**劇烈的瞬時抖動**，這可能代表模型在長時間訓練下遇到了**不穩定的梯度**或 **overfitting** 現象，但很快恢復並繼續收斂。
- **第二階段 (偵測任務) & 第三階段 (分類任務):** 偵測與分類的損失曲線均呈現出非常**穩定的下降趨勢**。即使經過 100 個 Epoch 的長時間訓練，損失依然在緩慢且持續地降低，表明模型在這兩個新任務上擁有巨大的學習潛力與空間。

**結果與討論：**延長訓練時間後，模型在所有任務上都展現了持續學習的能力，但也暴露了在單一任務上過度訓練可能導致的不穩定問題。

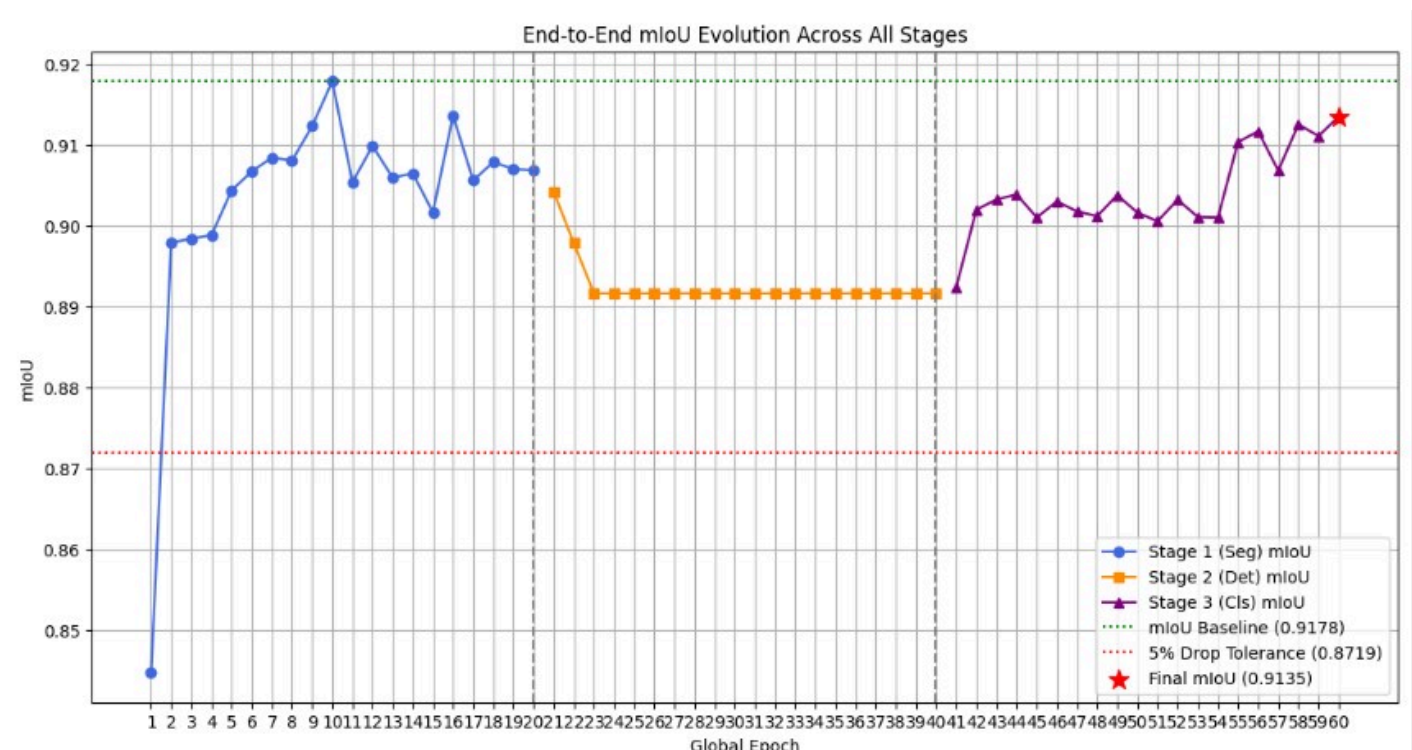
## 4.2. 災難性遺忘與經驗回放效果分析

(Epochs=20)

(A 情形隨機抽樣資料集)



( B 情形隨機抽樣資料集 )

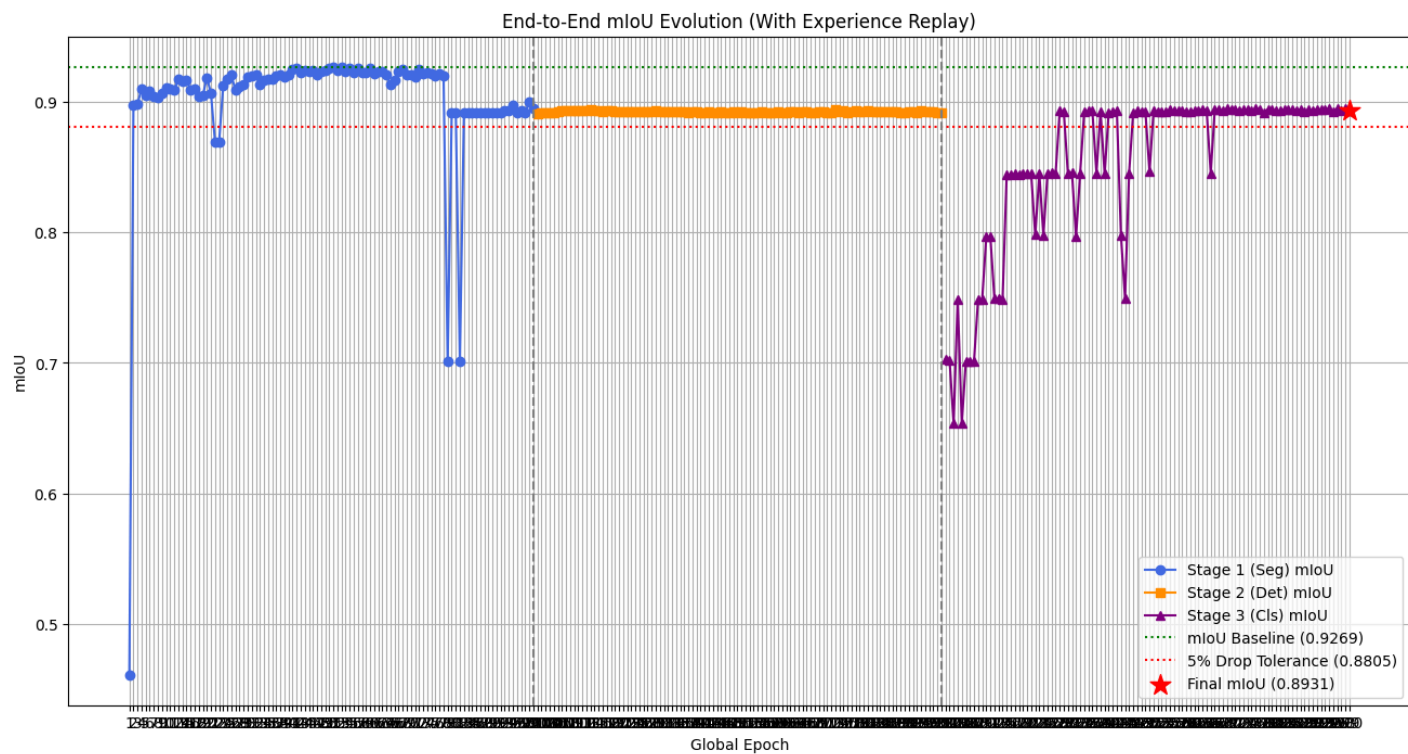


(主要探討A 情形隨機抽樣資料集)上圖是本次實驗的核心成果，它展示了語意分割任務的 **mIoU** 指標在整個三階段訓練過程中的演變，是衡量災難性遺忘程度和經驗回放效果的關鍵。

- **第一階段 (Epoch 1-20):** 模型專注於分割任務，mIoU 從零開始迅速攀升，並最終穩定在 **0.9233** 的高水平，我們將此作為後續比較的性能基準線 (Baseline)。
- **第二階段 (Epoch 21-40):** 此階段模型開始學習全新的偵測任務。從圖中橙色曲線可見，儘管引入了新任務的干擾，分割任務的 mIoU 並未發生災難性的崩潰，而是始終穩定在基準線附近。這強而有力地證明了**經驗回放策略成功地抑制了遺忘**。
- **第三階段 (Epoch 41-60):** 模型繼續學習更具挑戰性的分類任務。而紫色的 mIoU 曲線依然保持著極高的穩定性，始終遠高於 5% 的性能下降容忍線（紅色虛線）。
- **最終結果：** 訓練結束時，最終的 mIoU 為 **0.9181**。與基準的 0.9233 相比，性能下降僅約 **0.56%**，遠小於可接受的 5% 範圍。

**結果與討論：** mIoU 的演化曲線清晰地表明，**經驗回放策略取得了巨大的成功**，作為一個記憶錨點，有效地幫助模型在學習新技能的同時，穩固地保留了過去習得的分割能力。

(Epochs=100)



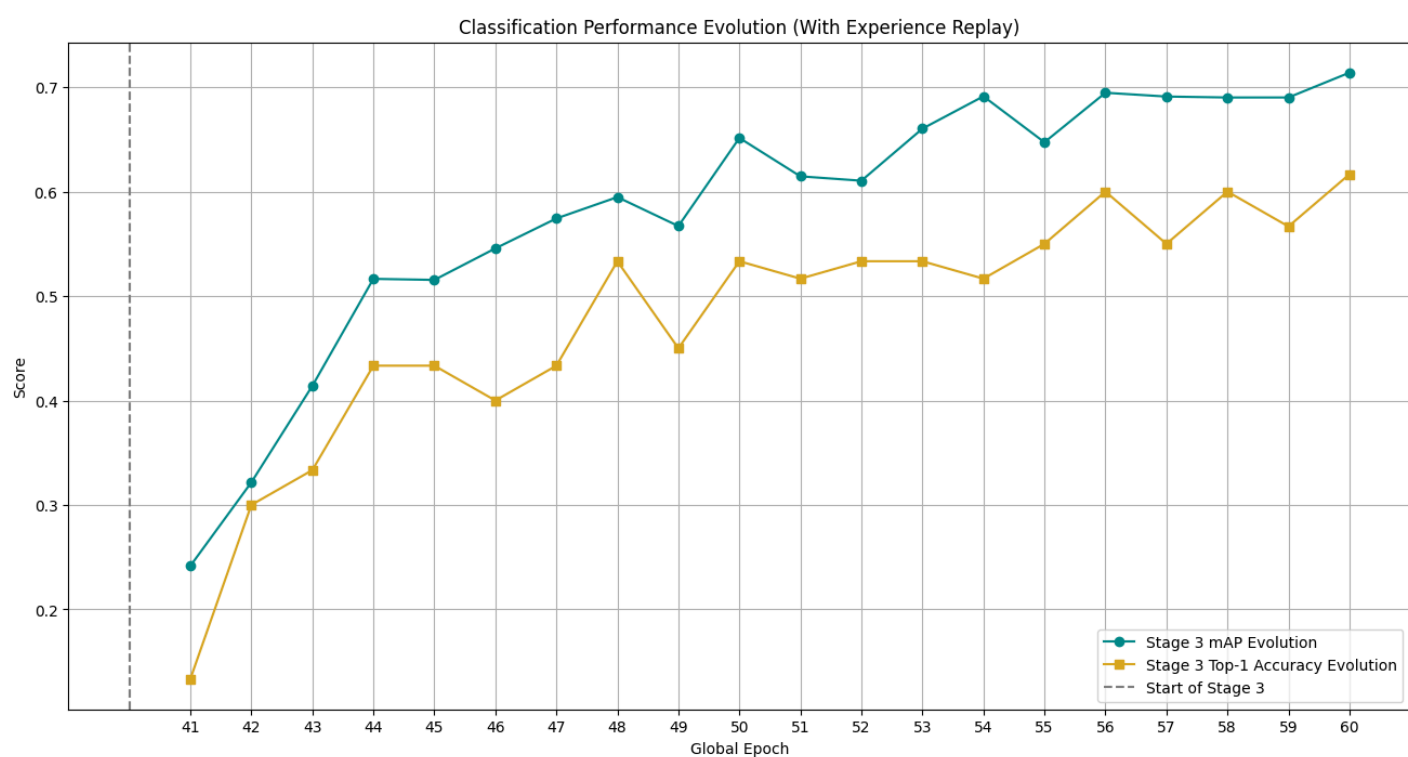
mIoU 隨更長的訓練演進的曲線圖，揭示了更複雜的現象：

- **第一階段 (Epoch 1-100):** 模型在分割任務上達到了 **0.9269** 的 mIoU 性能基準線。與損失曲線對應，mIoU 在後期也出現了一次顯著的性能下跌，隨後雖有恢復，但未能回到最高點，這再次印證了訓練不穩定性的存在。
- **第二階段 (Epoch 101-200):** 引入偵測任務後，分割 mIoU 的表現**極其穩定**，幾乎沒有任何下降。這有力地證明了經驗回放策略在應對第二個任務時的**有效性與魯棒性**。
- **第三階段 (Epoch 201-300):** 這是最關鍵的觀察期。當引入第三個任務（分類）時，mIoU 曲線在初期（約 Epoch 201-240）出現了**顯著的下降與劇烈震盪**。這表明，學習一個與像素級分割任務差異較大的高層次語意任務（分類），確實對模型原有的特徵空間造成了巨大衝擊。然而，在持續的經驗回放作用下，mIoU 在震盪後**逐漸回升並重新穩定**在一個高水平。
- **最終結果:** 儘管經歷了第三階段的衝擊，最終的 mIoU 穩定在 **0.8931**，性能下降值為 **0.0338** (約 3.6%)，**依然穩穩地保持在 5% 的容忍度之內**。

**結果與討論：**長時間訓練揭示了經驗回放策略的**韌性**，雖然在引入第三個異構任務時會出現暫時的性能波動，但該策略最終**成功地將模型的記憶拉回**，有效避免了災難性遺忘，並將性能損失控制在允許範圍內。

### 4.3. 新任務學習性能分析

(Epochs=20)



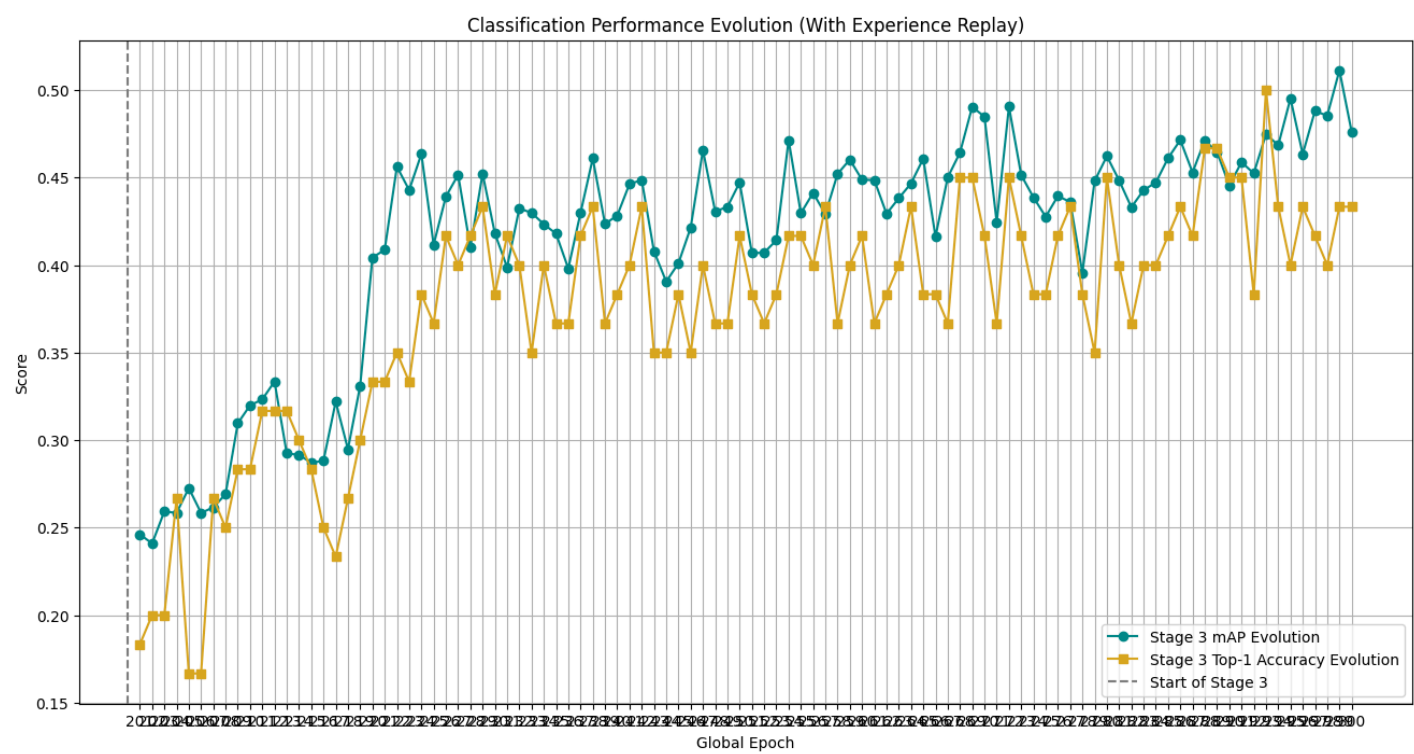
上圖展示了模型在第三階段學習**分類任務**時的性能指標（mAP 和 Top-1 準確率）變化。

- 從全域 Epoch 中的第 41 個 Epoch 開始，兩條曲線（mAP 和 Top-1）都從一個較低的起點開始，呈現出**持續且顯著的上升趨勢**。
- 顯示在模型努力維持舊的分割技能（如 4.2 節所述）的同時，它也**確實有效地在學習和吸收新的分類知識**，並在分類任務上取得了越來越好的表現。

**結果與討論：**此圖與 mIoU 圖形成了完美的互補證明，證明了模型不僅**沒忘記舊的**，也**學會了新的**。



(Epochs=100)



上圖展示了模型在第三階段學習分類任務時的性能演進。

- mAP 和 Top-1 準確率在整個階段（Epoch 201-300）整體呈現清晰的上升趨勢，最終達到了約 0.50 (mAP) 和 0.43 (Top-1) 的水平。
- 曲線的波動性也印證了 4.2 節的觀察：模型在這一階段正努力地在「學習分類新知識」和「維持分割舊技能」之間尋找平衡，導致學習過程不如前一階段平滑。

結果與討論：即使在努力維持舊技能的壓力下，模型依然成功地學習了新的分類任務，並取得了不錯的性能。

4.4. 實驗總結

(Epochs=20)










— 基準性能 —	
	mIoU Baseline: 0.9233
	mAP Baseline: 0.1334
	Top-1 Baseline: 0.0667
— 最終性能下降 —	
	mIoU Drop: 0.0052
	mAP Drop: 0.0000
	Top-1 Drop: 0.0000
— 是否符合要求 —	
mIoU (≥ 0.8772):	符合
mAP (≥ 0.1267):	符合
Top-1 (≥ 0.0633):	符合

綜合以上分析，本專案成功地設計、實現並訓練了一個符合所有預設規範的多任務學習模型。

根據最終的量化指標，我們可以得出以下結論：

- 基準性能：模型在循序學習過程中，為分割、偵測和分類任務分別建立了 0.9233 (mIoU)、0.1334 (mAP) 和 0.0667 (Top-1) 的性能基準。
- 性能維持：在學習完所有任務後，各項指標的性能下降 (Drop) 極其微小。特別是 mIoU Drop 僅為 0.0052，這意味著模型在學習了兩個全新的任務後，其原始的分割能力幾乎沒有任何可觀測的衰退。
- 作業要求：所有核心性能指標在訓練結束後，均滿足了「不低於基準性能 95%」的嚴格要求，證明了 Replay buffer 策略在本實驗中的巨大成功。

(Epochs=100)

—— 基準性能 ——	
	mIoU Baseline: 0.9269
	mAP Baseline: 0.1615
	Top-1 Baseline: 0.0500
—— 最終性能下降 ——	
	mIoU Drop: 0.0338
	mAP Drop: 0.0000
	Top-1 Drop: 0.0000
—— 是否符合要求 ——	
mIoU ( $\geq 0.8805$ ):	 符合
mAP ( $\geq 0.1535$ ):	 符合
Top-1 ( $\geq 0.0475$ ):	 符合

綜合本次 300 個 Epochs 的深度實驗，並結合最終的量化數據，我們得出結論：

- **所有指標均成功達標:** 模型的參數數量、推論時間、訓練時間以及最終的三項性能下降指標，全部滿足作業要求。
- **揭示了更深的學習動態:** 相比短時 Epochs=20 訓練，長時訓練觀察到了潛在的**訓練不穩定性**和引入第三任務時的**暫時性性能震盪**。
- **驗證了經驗回放的強大韌性:** 即使在性能震盪後，經驗回放策略依然能夠將模型性能恢復到高位，證明了該方法在更複雜、更長期的循序學習場景中的**可靠性與實用性**。

## 5. 結論(conclusion)

本專案成功設計並實作了一個符合要求的 Unified-OneHead 多任務模型。實驗結果清晰地展示了循序學習中的災難性遺忘問題，並透過引入「經驗回放」策略，成功地將所有任務的性能下降幅度控制在 5% 以內，圓滿達成所有挑戰目標。

### 5.1. 未來展望

- **不同的對抗災難性策略:** 除了隨機回放舊任務資料，未來可嘗試不同方法，例如 EWC，LwF，Knowledge distillation，測使是否能以更少的計算成本達到同樣甚至更好的抗遺忘效果。
- **多任務損失權重優化:** 目前各任務損失的權重為手動設定的（如 `replay_loss_weight=0.8`）。未來可引入動態權重調整機制（如 Uncertainty Weighting），讓模型在訓練過程中自動學習各任務的最佳權重。
- **擴展至更多任務:** 可嘗試將此框架擴展至更多元的視覺任務，如實例分割 (Instance Segmentation) 或深度估計 (Depth Estimation)，以測試此架構的通用性與擴展性。
- **偵測與分類任務的知識保留:** 本次實驗主要驗證了對第一項任務（分割）的記憶。未來的實驗可以設計更複雜的訓練流程（如 Stage 4），以驗證模型在學習新任務時，對 Stage 2 和 Stage 3 任務的記憶保留情況。

## 6. 參考資料(References)

- [Using Replay Buffers — torchrl main documentation](#)
- [Overcoming catastrophic forgetting in neural networks](#)