Lab05 Report – Deep Learning

學生：梁皓鈞，學號：314580042，系所：前瞻半導體研究所晶片組

一、 Task 1

(一)、 Model Architecture Design and Improvements

1. 模型選擇分析

在 Lab 5 Task 1 中，我的首要目標是實作高效能 Semantic Segmentation 模型，並確保 mIoU 指標能突破 0.72 的門檻。為此，我在初期架構探索階段深入評估了幾種主流模型。

首先研究語意分割先驅 FCN，儘管其開創了端到端訓練先河，但實驗數據顯示其基於 Summation 的 Feature Fusion 機制往往導致預測邊緣粗糙，難以滿足高精度分割需求。接著，我評估了以 Atrous Convolution 為核心的 DeepLab 系列，雖然該系列透過 ASPP 類別強化了 Global Context 捕捉能力，但權衡模型複雜度與運算資源的 Trade-off，以及本次任務對 Pixel-Level 定位精度的嚴格要求，DeepLab 並非現階段最佳選擇。

最終我決定採用經典 U-Net 架構，其關鍵優勢在於獨特的 Encoder-Decoder 對稱結構，特別是 Feature Map Concatenation 機制。此設計能在有限資料量下有效結合深層抽象語義與淺層高解析度細節，確保分割結果在維持語義正確性的同時，亦能保有清晰且精確的幾何輪廓。

2. 對 U-Net 架構的現代化優化

我在 network.py 中的實作並未完全照搬 U-Net 原始論文架構，而是結合現代深度學習的技術進行了多項改良。在核心 Double Conv 類別設計方面，我延續了原始 U-Net 連續堆疊的思路，採用類似 VGG 的雙 3x3 卷積設計，藉此在維持較少參數量的前提下，獲得等效於 5x5 卷積的 Receptive Field。此外，我與原始論文最大的差異在於 Padding 的處理細節。原始 U-Net 架構因未做 Padding 導致 Feature Map 尺寸隨層數加深而逐漸縮小，但我將 Padding 參數設定為 1，即採用 Same Padding。這項改動能確保 Feature Map 經卷積操作後，空間維度能保持不變，其不僅大幅簡化後續 Skip Connections 對齊形狀的複雜度，更有效地避免了邊緣資訊的流失。

3. 訓練收斂能力優化與記憶體管理優化

為了進一步優化模型收斂性能，並緩解 Internal Covariate Shift，我在每一層卷積後均加入了原始 U-Net 所未具備的 nn.BatchNorm2d。此項改進不僅平滑了 Loss Surface 以加速訓練過程，更起到輕微的 Regularization 作用，有助於避免 Overfitting。而在 Activation Function 的選擇上，我採用了啟用 Inplace 參數的 ReLU，藉由其 In-place 特性在處理高解析度輸入時直接於記憶體原址進行運算，從而顯著降低 VRAM 需求並優化整體 Memory Access Efficiency。

4. Down Sampling Path 設計優化

在負責下採樣的 Down Sampling Path 中，我在 Down 類別內整合了 kernel size 為 2 的 nn.MaxPool2d。相較於加權混合局部資訊的 Stride Convolution，Max Pooling 能直接選取 Local Maxima 以避免 Background Noise 稀釋特徵訊號，藉此更有效地保留了諸如物體邊緣等強度較高的特徵，並引入一定程度的 Translation Invariance。針對深層特徵的配置，我權衡了 Model Capacity 與 Computation Cost，在 down4 階段將輸出通道數限制在 512 而非傳統的 1024；此舉旨在於模型容量與推論速度之間取得平衡，避免在 Bottleneck 處產生過大的運算瓶頸。

5. Up Sampling Path 設計優化與動態對齊優化

原始 U-Net 架構嚴格限制輸入尺寸為一大缺點，為了增強模型穩健性，我採用具可學習權重的 nn.ConvTranspose2d 取代固定的 Bilinear Interpolation 來恢復特徵圖尺寸，使網路能依據 Loss Function 學習最佳上採樣策略。此外，為了妥善處理上採樣特徵與 Skip Connection 特徵間可能存在的 Dimension Mismatch，我不預設兩者尺寸能完美匹配，而是實作 Dynamic Padding 機制；藉由計算兩者在高度 diffY 與寬度 diffX 上的差異並利用 F.pad 函數來做 Dynamic Padding，確保了使用 torch.cat 拼接時能完全對齊形狀，徹底解決奇數輸入尺寸所導致的錯位問題，大幅提升模型對任意輸入尺寸的適應性。

6. 細節模型架構

在基礎類別方面，我首先實作 Double Conv 類別，其內部利用 nn.Sequential 串接兩組相同的結構，這兩組結構皆包含 kernel size 為 3 且 padding 數為 1 的 Conv2d 層、BatchNorm2d 層以及設定 Inplace 為 True 的 ReLU。接著是負責下採樣 Down 類別的實作，我同樣使用 nn.Sequential 串接一個 kernel size 為 2 的 MaxPool2d 層與前述的 Double Conv 類別。

相較於 Down 類別，負責上採樣的 Up 類別設計更為細緻。在初始化階段，我採用 kernel size 與 stride 皆為 2 且輸入輸出通道數減半的 ConvTranspose2d 放大特徵圖，後續銜接標準 Double Conv 進行 Feature Fusion。針對 forward 流程，Up 類別接收來自下層的 x1 與 Encoder 跳躍連接的 x2。首先對 x1 執行反卷積上採樣，為避免 Dimension Mismatch 並提升模型對任意尺寸的適應性，我利用 F.pad 計算 x2 與 x1 在長寬上的差異值後，對 x1 施加 Edge Padding。待確保 x1 與 x2 形狀完全對齊後，便能在維度 1 透過 torch.cat 拼接兩者，最後送入 Double Conv 完成 Feature Fusion。

主體 U-Net 類別在初始化時預設接收 3 個輸入通道與 8 個輸出類別，整體流程區分為 Encode 與 Decode 兩部分。Encode 路徑始於 InC 層，利用 Double Conv 將輸入轉換為 64 個通道，接著依序經過 Down1 至 Down4 四個下採樣階段，通道數分別變化為 128、256 與 512，其中在 Down4 階段輸入與輸出皆維持 512 個通道來避免過大的計算瓶頸。Decode 路徑則包含 Up1 至 Up4 四個上採樣階段，其中 Up1 接收 1024 個合併通道並輸出 256 個通道，Up2 接收 512 個

通道並輸出 128 個通道，Up3 接收 256 個通道並輸出 64 個通道，Up4 接收 128 個通道並輸出 64 個通道。最終透過 OutC 層。即一個 kernel size 為 1 的 Conv2d，將 64 個特徵通道轉換為對應 n_classes 的輸出 logits。此外，我也提供了一個 load_model 函式，用於正確建立一個預設參數的 U-Net 實例，並載入指定路徑的 state_dict 權重檔至模型後，回傳該模型物件。

7. 結論與優缺點分析

總結來說，此架構設計的優點在於具備強大的特徵還原能力，並由對稱結構帶來 Gradient Flow 穩定性以及動態 Padding 機制所賦予的穩健性。然而其缺點亦顯而易見，Transposed Convolution 引入了額外參數量，加上 Skip Connections 倍增特徵通道，致使 Decoder 端的卷積運算須承受極高的 FLOPs 與顯存壓力。這些計算成本與資源開銷將是我後續於 Task 2 中必須透過量化技術來解決的重要瓶頸。

(二)、 Accuracy Improvement Strategies

1. Loss Function

針對 Task 1 處理 8 種類別的 Segmentation 特性，在 Loss Function 配置上我選用標準的 Cross Entropy Loss。此函數能有效衡量預測概率分佈與真實標籤分佈間的差異，並於像素級別對錯誤分類進行懲罰，促使模型學習更精確的邊界判定。同時 Cross Entropy Loss 具備良好的梯度傳遞特性，能有效維持數值穩定性並大幅降低深層網路常見的 Gradient Vanishing 風險，確保誤差能順利回傳至前端特徵提取層。

2. Optimizer

在確立以 U-Net 為骨幹的架構後，主要目標在於同步提升 Accuracy 並優化 Loss 收斂情形。實驗配置助教將 Input Size 設定為 256x512，我設置 Batch Size 為 16，藉此降低記憶體開銷並確保梯度估計的準確性。考量到語義分割任務伴隨著龐大的參數空間及深層網路梯度尺度差異，優化器選用 Adam 並將初始 Learning Rate 設為 0.002。由於 Adam Optimizer 結合了 Momentum 的 First Moment 與 RMSProp 的 Second Moment 估計，具備為各參數自適應動態調整 Learning Rate 的核心優勢，相較於傳統 Stochastic Gradient Descent 算法，應用於 U-Net 深層架構時不僅收斂速度更快，更能有效避免訓練初期的 Loss 震盪。檢視訓練歷程，模型在 Epoch 0 時 Training Loss 為 0.7292，Validation mIoU 為 0.5007；隨著 Adam 引導梯度下降，才到 Epoch 5 時，Loss 便迅速收斂至 0.3543，mIoU 亦大幅躍升至 0.6448，顯示此 U-Net 架構具備優異的 Feature Extraction 能力且 Adam 具有強大的訓練收斂能力。

3. Overcome Overfitting

為了提升 Generalization Capability 並優化收斂過程，本實驗導入 StepLR Scheduler 動態調整 Learning Rate。初期先利用較大的 Learning Rate 進行廣泛 Exploration 以避免過早陷入 Sub-

optimal Local Minima，隨後人工尋找於第 20 個 Epoch 將 Learning Rate 衰減 0.5 倍進入細緻精修階段。此策略使模型在後期能以更小的步伐微調權重，降低在 Loss Surface 最低點附近的震盪並穩定收斂。此外為防止 Over-training，我搭配 Model Checkpointing 機制監控 Validation mIoU 以自動儲存最佳權重。數據顯示在 Epoch 20 觸發 StepLR Decay 後收斂精度顯著提升，並於 Epoch 24 取得 Training Loss 0.1946 與 Validation mIoU 0.7607 的最佳表現，證實 Adam Optimizer 配合 Cross Entropy Loss 與 StepLR Scheduler 能有效平衡收斂速度與最終性能。

I. Task 2

i. Methods for Lowering Complexity and Bridging the Accuracy Gap

Lab 5 Task 2 的核心挑戰在於權衡 Complexity 與 Accuracy，目標旨在降低 FLOPs、Parameters，並且在守住 mIoU 大於 0.66 門檻的同時，最小化下述 FoM 指標：

$$\text{FoM} = (\text{FLOPs})^2 \times (\text{Model Size})^2 \times (\text{mIoU} - 0.65)^{-1}$$

為了在資源受限的場景下輕量化 Task 1 模型，我重構 network.py 並全面採用 Depthwise Separable Convolution 來取代標準的 Convolution。實作上，我將 nn.Conv2d 的 Groups 設置為輸入通道數，來執行專注於 Spatial 特徵的 Depthwise Convolution，其後串接執行專注於 Channel 特徵的 Pointwise Convolution；這種分解方式在 Kernel Size 為 3x3 的情況下，能大幅縮減運算量至原先標準卷積的 1/9。另外，在透過此輕量化類別實作 Efficient U-Net 時，我激進地將 Channel 數逐步縮減至 9；此舉雖然極大地壓縮模型的 Parameters 與 FLOPs，但也導致模型的表達能力顯著下降。為了補償 Channel 數的縮減所帶來的資訊損失，我將 Efficient Channel Attention 整合進 Efficient Double Convolution 內，有別於傳統 SE Network 的降維破壞了 Channel 間的對應關係，ECA 可透過：

$$k = \left\lfloor \frac{|\log_2(\text{Channels}) + b|}{\gamma} \right\rfloor$$

搭配參數 γ=2、b=1，來自適應地計算出 Kernel Size，並以 nn.Conv1d 實現不降維的 Local Cross-Channel Interaction。此外，考量到後續的量化需求，我在實作 forward pass 時，特別採用 nn.quantized.FloatFunctional 的 mul 方法來執行 Attention Map 加權；此舉不僅保證了模型在數值精度轉換過程的穩定性，同時也維持參數量在線性級別，實現高效率的特徵增強。

針對 Efficient Up 類別的架構優化，我主要採取兩個策略：首先，為了解決 Task 1 中 Transposed Convolution 所造成的 Checkerboard Artifacts，我改以 scale_factor = 2 且 mode = 'bilinear'的 nn.Upsample 來取代 Transposed Convolution；此改動利用固定的算法直接消除 Checkerboard Artifacts，同時也省去額外的參數學習成本。其次，我改良了 Skip Connections 的融合機制：將傳統 Concatenation 替換為 Additive Skip Connection 以降低頻寬消耗。實作上，我利用兩個

1x1 卷積層將特徵投影至相同輸出 Channel 數後，透過 FloatFunctional 的 add 方法進行結合；此舉不僅最小化 Decoder 的 Memory Bandwidth，FloatFunctional 也能直接支援後續 QAT 流程，有效解決一般加法算子在量化時常遇到的相容性問題。

為了彌補模型輕量化所造成的精度損失，我實作 Composed Loss 來進行優化。首先，以權重 1.0 的 Cross Entropy Loss 維持基礎像素分類準確度，並搭配權重 1.0 的 Dice Loss 來解決類別不平衡的問題。接著，為了進一步拉升指標，我引入了權重 0.5 的 Lovasz Softmax Loss；它是基於 Jaccard Index 的凸替代函數，能透過誤差排序讓模型直接針對 mIoU 進行梯度優化，有效地解決訓練目標與最終評估指標不一致的問題。

針對量化的策略，考量到 Post Training Quantization 在極輕量模型易造成精度顯著下降，我改採完整的 Quantization Aware Training。實作配置上，我先將 Backend 指定為 qnnpack，並透過 prepare_qat_model 插入 QuantStub、DeQuantStub 及 Fake Quantization 節點。進入訓練階段後，為讓模型適應低精度的權重分佈，我將 Learning Rate 降至原參數的 0.1 倍，即：3.75×10^{-4}，並進行 35 個 Epoch 的微調。最終經由 convert_to_quantized 轉出 INT8 的模型 Warpper，部署於 CPU 做推論，實驗顯示 mIoU 達到 0.668445，不僅成功跨越 0.66 門檻，更保有 0.008 的 Margin，驗證了 QAT 在輕量架構下的可行性。

ii. Justification and Implementation Details

本設計在模型架構的決策與實作上，嚴謹參考了理論與實驗數據。首先，在 Backbone 方面我選用前述的 Efficient U-Net 架構，其經過多次實驗定案在能最小化 FoM 指標的 base channels 設定：我將最終的 base channels 數值定在 9，形成 9、18、36、72 的輕量化通道擴張架構。

接著，為了最大化計算效率，我引入 MobileNet 思維的輕量級 Depthwise Separable Convolution；PyTorch 實作細節為：我將標準 Conv2d 的 Groups 參數設定為輸入通道數便可完成 Depthwise Convolution。在 Depthwise Convolution 後，緊接著的便是執行 Pointwise Convolution，以達成通道間的特徵融合；此舉顯著地降低模型運算複雜度。另外，為確保數值穩定性並引入非線性，Depthwise 與 Pointwise 卷積層後皆串接了 BatchNorm2d 與 ReLU；在維持特徵提取精度的同時，確保模型性能。

考量到前述通道數縮減導致了特徵資訊的流失，我在 Efficient Double Conv 架構中引入 ECA Block，其利用此輕量級 Attention 機制來強化輕量化模型的特徵表達能力；PyTorch 實作細節為：首先，我採用 Adaptive Average Pooling 壓縮全域空間資訊，隨後依據通道維度並搭配 γ=2、b=1 來動態計算 Kernel Size，以執行 1D Convolution，捕捉通道間的局部相依性，再透過 Sigmoid 函數生成通道權重。此後，為了將生成的權重套用於原始特徵圖，我需要進行逐元素的乘法運算，針對此步驟，我特別將該加權操作封裝於 nn.quantized.FloatFunctional 類別中；主要目的是確保在進行 Quantization Aware Training 流程時，它能正確觀測並統計該操作的數值動態

範圍，有效預防模型在量化過程中所可能發生的精度問題。

Decoder 的架構設計上，我在 Efficient Up 類別中特別選用 Bilinear Interpolation 來進行上採樣，主要是看重其無參數特性，又可同時提供平滑輸出。緊接著，為了處理特徵融合階段常見的 Dimension Mismatch，我引入了 skip_proj 與 up_proj 這兩個 1x1 卷積層，分別針對來自 Encoder 端 Skip Connection 的特徵圖，以及剛完成上採樣的特徵圖，來對齊 Channel 維度。此外，考量到前層的 Max Pooling 操作在碰到奇數尺寸輸入時，容易導致 Spatial Dimension 發生 Mismatch，我在進行融合前，特別加入了一套基於 F.pad 的 Dynamic Pooling 機制，確保來自 Encoder 端 Skip Connection 的特徵圖，以及剛完成上採樣的特徵圖，其長寬能精確吻合，從而避免 Dimension Mismatch。在完成上述對齊步驟後，我接著採用 nn.quantized.FloatFunctional 來執行逐元素加法運算，以實現特徵聚合。儘管此設計策略相較於使用 Transposed Convolution 會捨棄掉部分的特徵學習能力，但是針對資源受限的應用場景而言，它是個能確保模型具備完整 INT8 量化相容性、又能確保模型在運算效能與模型複雜度間取得平衡。

進入模型優化階段，考量到模型極低參數量的限制，僅依靠 Cross Entropy Loss，往往難以讓模型收斂出理想的 mIoU Performance。為了突破此瓶頸，我引入了直接針對 Jaccard Index 進行優化的 Lovasz Softmax Loss，藉此強化對區域重疊度的評估能力；同時，為改善類別樣本不均的問題，我進一步整合了 Dice Loss。最後，在最終的 Composed Loss 設計上，我採取複合式加權，將 Total Loss 定義為 1.0 倍的 Cross Entropy Loss 加上 1.0 倍的 Dice Loss 以及 0.5 倍的 Lovasz Softmax Loss。透過這套多重損失函數的加權組合，此輕量化模型便能夠在維持 Pixel-Level 分類準確度的同時兼顧類別平衡，有效最佳化全域指標。

進入量化流程。首先運用 torch.quantization.QuantStub 與 DeQuantStub 來精確界定 Computation Graph 的量化範圍，接著為了在訓練階段就能模擬量化模型行為，並確保最終部署效能，我透過 prepare_qat_model 將推理後端配置為專用的 qnnpack。此外，考量到在低精度環境下，標準加法運算算子容易引發數值不穩定或精度損失，我對所有的殘差連接與特徵融合層強制導入 FloatFunctional 類別來取代傳統加法；此舉確保了計算過程的穩定性，並有效規避了 INT8 運算潛在的數值溢出風險。

為提升模型在助教部署時的穩定度，我實作了一套穩健的 load_model。這個函數會先檢查 State Dict 內容，自動判斷模型是屬於 FP32 還是 INT8。若識別為量化模型，系統將進一步過濾掉 Profiling 階段殘留的 Redundent Key，例如：total_ops 與 total_params，以確保權重檔的整潔。在載入權重的過程中，我將 strict 參數設定為 False 啟用非嚴格模式，藉此容許些微的鍵值差異。最後，為防止 CPU 專用的量化模型因為被誤用.to(device)指令而崩潰，我將模型封裝在自定義的 Quantized Model Wrapper 中，攔截這類操作錯誤。

實驗數據充分驗證本策略的有效性。首先在 FP32 預訓練階段，藉由導入 Composed Loss 優化了模型的收斂行為，使模型在第 44 個 Epoch 即達到 0.680889 的 mIoU，表現顯著優於預期目

標。有了良好的浮點數模型基礎後，其便接著進入 35 個 Epoch 的 QAT 微調，為確保權重在量化模擬過程中能穩定更新，我特別將 Learning Rate 調降至初始值的 0.1 倍。在此設定下，模型於 Fake Quantization 環境中仍能維持 0.675107 的 mIoU，顯示參數已成功適應量化所引入的 Noise。最後，我利用 convert_to_quantized 將模型正式轉換為 INT8 格式並部署至 CPU 端執行推理，此時模型參數與 Buffer 皆已壓縮為 8-bit 整數，實測 mIoU 為 0.668445。此結果雖相較於原始 FP32 版本有微幅精度折損，仍成功達到 0.66 的門檻，證實結合架構輕量化與 QAT，確實能有效突破資源受限 Semantic Segmentation 任務的效能瓶頸。