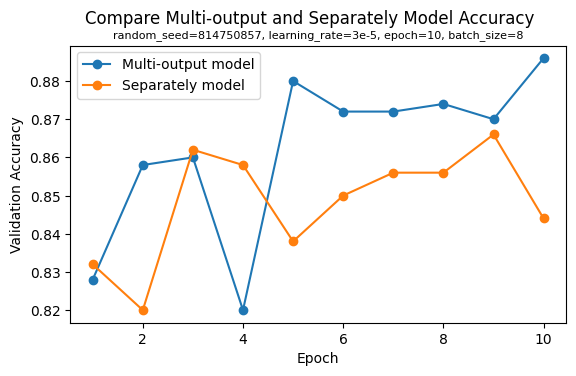
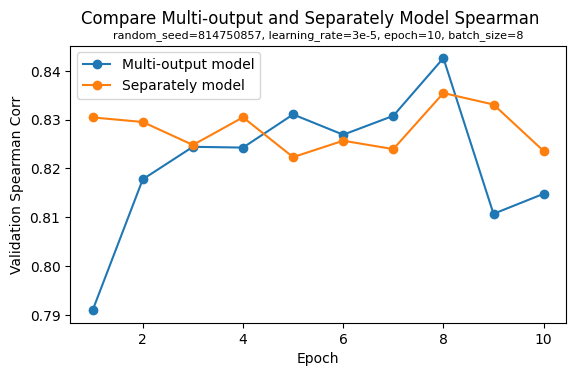
**11310CS563100 NLP Assignment 3: Multi-output learning Report**

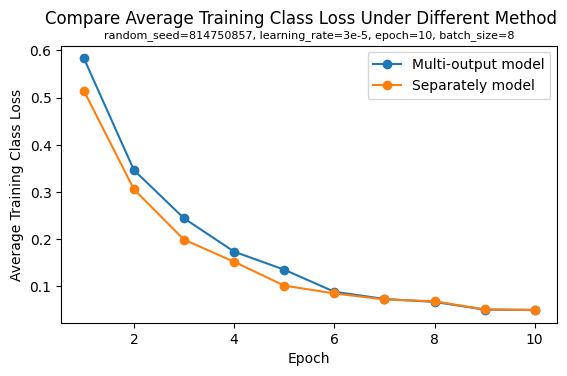
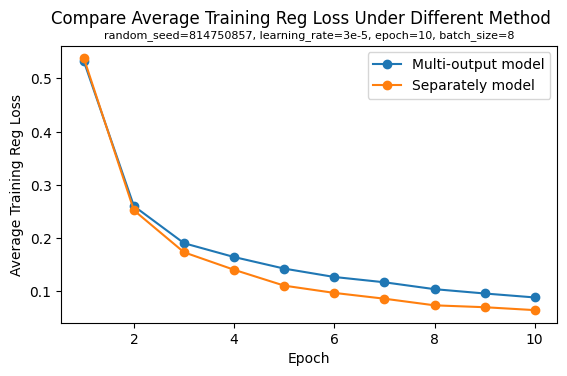
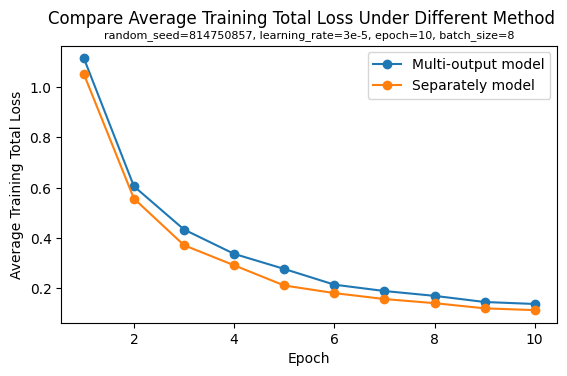
NTHU 110020007 施淙綸

* Which (pre-trained) model do you use? Why to choose the model?
  + 我使用最原始的 "google-bert/bert-base-uncased"而非其他fine-tuned過的bert-base model，不是因為表現或效能的原因，而是原始BERT通用程度最高（當然他表現的也還不錯）。而且不太確定其他fine-tuned model是用什麼方法訓練或優化的，如果和我實作的相近的話有機率會overfit，可能會需要其他技巧來解決，或是只用更少的epoch，但這樣就和作業要我們練習訓練模型的目的相違背了。
* Compared with models trained separately on each of the sub-task, does multi-output learning improve the performance?
  + 我一開始認為multi-output learning訓練出來的output應該表現比較差，畢竟是使用相同的特徵作為input，再各自進一步訓練全連接層對應回歸和分類任務。於是我訓練一個最單純的multi-output model (Model 1)，和兩個BERT的separately model (Model 6)，模型表現如下：

一張含有 文字, 行, 圖表, 繪圖 的圖片

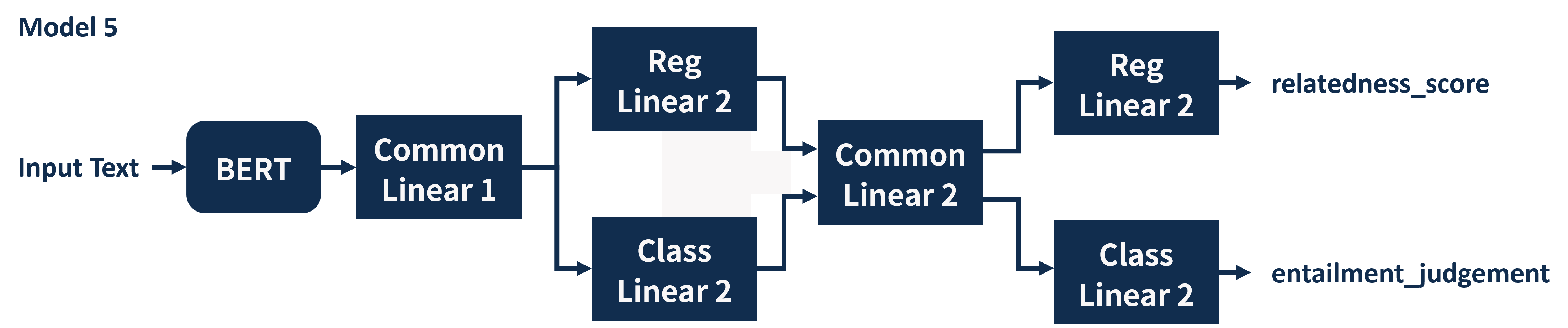
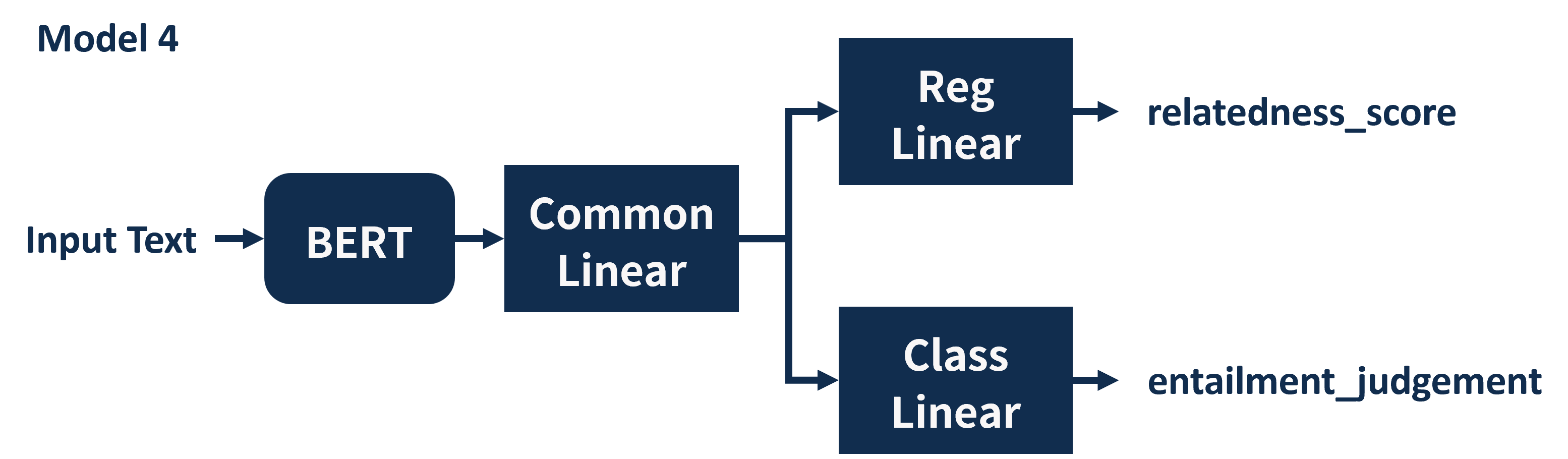
自動產生的描述

* + 從三個performance 可以看到，基本上multi-output的表現從第五代開始都比separately來的好（除了multi-output在spearman第九代和第十代較低外，推測在訓練更多代數之後仍然可以超越separately）。在得出數據之後我思考了一下，使用相同特徵訓練作為input來訓練不同任務在其他資料集的情況很常見，所以其實並不會直接讓模型表現很差。另外在訓練的過程中模型也在調整裡面萃取特徵的方式，來同時滿足兩種任務的需求。
  + 而在Training Loss的部分，兩種都是逐步下降的趨勢。而multi-output在total loss都較separately高，但剛才的performance顯示前者表現反而略高一些。分別來看兩種任務的loss可以發現multi-output在classification task的loss和separately相當或更低，在regression task的loss一直追不上separately。



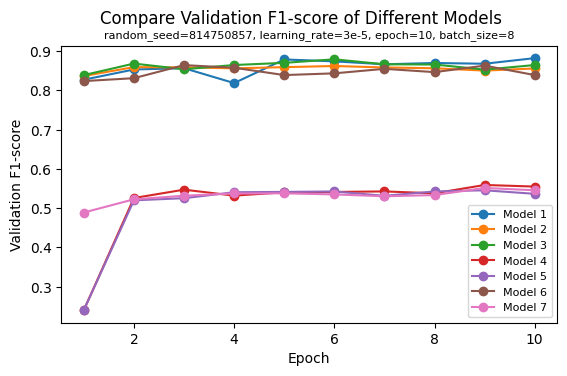
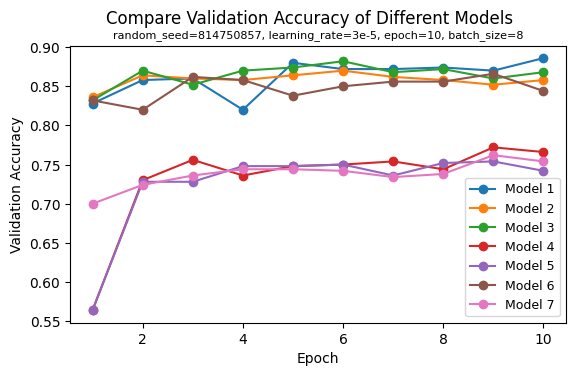
* + 除了模型表現之外，在訓練時長、耗費的資源和RAM大小上，multi-output model有絕佳的優勢，因為separately model每次都使用兩個BERT、調整兩個BERT參數，計算量和佔用記憶體都是前者的兩倍。
* How do you improve your model performance?
  + 這次的訓練任務比較能變動的部分是multi-output在BERT後，針對兩種任務的全連接層的架構，因此我總共訓練了七種模型，每一種呈現不同的模式和架構：

|  |  |  |
| --- | --- | --- |
|  | BERT模式 | 全連接架構 |
| Model 1 | Multi-output | BERT後各自一層Linear |
| Model 2 | Multi-output | BERT後各自二層Linear，加入ReLU |
| Model 3 | Multi-output | Model2 + dropout 0.1 |
| Model 4 | Multi-output | Model3 + common layer |
| Model 5 | Multi-output | Model4 + another common layer |
| Model 6 | Separately | 兩個BERT和各自的Linear |
| Model 7 | Multi-output | Model5 + 更多維度的common layer 2 |



* + 上方圖片分別代表Model 4和Model 5的架構圖，兩個model都是新增共同的全連接層，期望會因此增加模型表現。實際各模型的performance比較如下：

一張含有 文字, 行, 螢幕擷取畫面, 圖表 的圖片

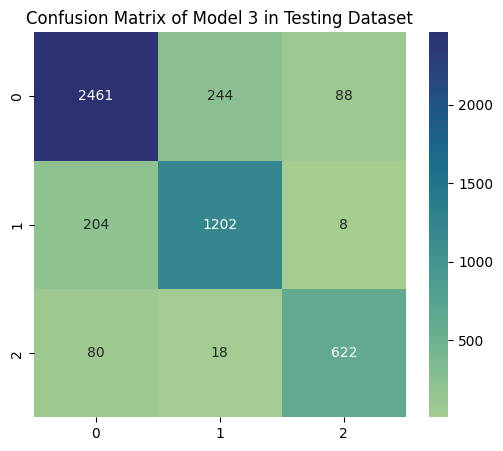
自動產生的描述

* + 由上圖左可以發現，在回歸任務上Model 1~3的震盪幅度較Model4~5, 7來的大，但總體而言這些模型在回歸任務大都得到的超過0.8的Spearman Corr。在大約第四代後，較多Linear層的Model 2表現較Model 1來的差，可能有些微的overfit（未記錄validation loss無法與training loss比較，因此只是推測），而加上Dropout layer的Model 3就大多比Model 2表現得來的好，這跟加入Dropout的目的相同（防止overfit）。
  + 在中、右兩圖的分類任務中可以看到，只要有堆疊共同的全連接層 (Model 4~5, 7)，表現都明顯不如傳統直接堆疊來的好。推測回歸任務因為是連續數值的關係，比較能根據不同情況fit出來，而分類任務因為是argmax的關係，比較難有大幅度的修正來變更他的預測值來增加模型表現。
  + Model 7和Model 5相比有更佳的表現是可預見的，因為在Common Layer 2有更多維度可以利用。
  + 和Assignment 2得到的經驗類似，在模型overfit之前，堆疊更多Linear層會逐步增加模型表現（Model 1 vs. Model 2）。我想learning\_rate的影響也相似，這次作業就沒有特別針對learning\_rate調整。
  + 綜上所見，加入共同的層數可能會導致特徵消失，限制模型對訓練任務的表現。也可以嘗試去除Model 5的Common Layer 1，我認為結果應該同樣不佳，不過礙於篇幅與時間就沒有特別訓練來觀察。
* Why does your model fail to correctly predict some data points? Please provide an error analysis.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Spearman Corr | Accuracy | F1-score |
| Model 1 | 0.8226 | 0.8764 | 0.8677 |
| Model 2 | 0.8164 | **0.8770** | **0.8709** |
| Model 3 | 0.8230 | 0.8697 | 0.8631 |
| Model 4 | 0.8187 | 0.7587 | 0.5485 |
| Model 5 | 0.8185 | 0.7420 | 0.5360 |
| Model 6 | **0.8269** | 0.8543 | 0.8454 |
| Model 7 | 0.8186 | 0.7548 | 0.5446 |

* + 上表各種模型在test dataset的表現數據，可以發現Model 2在回歸任務中得到0.8164，在分類任務是所有模型中最佳的；而加入Dropout Layer的Model 3則是在回歸任務表現第二（multi-output model第一），在分類任務排名第三。考量兩模型在兩種任務的綜合表現，我會選擇Model 3作為最終訓練出來的模型，所以接下來error analysis都是分析Model 3的表現。

一張含有 文字, 螢幕擷取畫面, 圖表, Rectangle 的圖片

自動產生的描述

* + 上圖是 Model 3在分類任務中的confusion matrix，可以發現模型把1分類成0和把0分類成1佔錯誤的大宗，其次是把2分類成0和把0分類成2，而1和2彼此錯誤分類的數量比起來極低（Testing: 1.79%, Validation: 0.2%）。換言之，從各標籤的意義來看（0中立，1包含，2矛盾），模型比較難分辨「包含、矛盾」與中立，而包含和矛盾對模型而言差距頗大，猜測在分類時有較明顯的特徵能夠將兩者分開。下表是部分分類錯誤的資料：

|  |  |  |  |
| --- | --- | --- | --- |
| premise | hypothesis | Truth | Pred |
| A brown dog is attacking another animal in front of the man in pants | A brown dog is helping another animal in front of the man in pants | 0 | 1 |
| A person in a black jacket is doing tricks on a motorbike | A person on a black motorbike is doing tricks with a jacket | 0 | 1 |
| The player is missing the basket and a crowd is in background | The player is dunking the basketball into the net and a crowd is in background | 2 | 0 |
| Two people are kickboxing and spectators are watching | Two people are fighting and spectators are watching | 1 | 0 |

一張含有 文字, 圖表, 螢幕擷取畫面, 繪圖 的圖片

自動產生的描述一張含有 文字, 螢幕擷取畫面, 圖表, 字型 的圖片

自動產生的描述

* + 上圖和下表是回歸任務的數據分佈，其中可以看到，總體而言訓練出來的模型和ground truth的分佈很相近，甚至連quantile的分佈都蠻相似的，可以合理推測模型有學習到資料的整體特徵，並能夠根據學習到的內容來預測回歸數據。而錯誤預測在回歸任務上我想觀察有點困難，考量作業規模和實作時間就分析到此。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Mean | SD | Min | Q1 | Q2 | Q3 | Max |
| relatedness\_score | 3.5300 | 1.0089 | 1.0000 | 3.0000 | 3.6000 | 4.3000 | 5.0000 |
| pred\_relatedness\_score | 3.6982 | 0.9864 | 1.0717 | 3.2931 | 3.7165 | 4.4996 | 5.0575 |

* 遭遇到的困難 & Reflections
  + 最開始計算Spearman Corr時沒有把ground truth和prediction value變成一致的形狀（前者 (batch\_size)，後者 (batch\_size, 1)），PyTorch有很貼心的提醒形狀不同算出來的值會不一樣，但我一直覺得他在危言聳聽，以為在這兩種形狀的情況下其實是一樣的，還想說為什麼Spearman Corr卡在0.76左右一直上不去。直到某次做了一個小實驗之後才發現他是對的，沒有變成一樣形狀的Spearman Corr和真實值相差超級多，於是我所有數據都重新跑過一次，所幸跑完之後模型表現就正常了。
  + 因為我的程式設計成所有需要數據的模型可以一起跑，一開始沒有留意到變數在jupyter notebook的垃圾回收和檔案執行不同，於是在堆積六個模型之後就開始跳RuntimeError: CUDA error: out of memory.，而且改成只跑一個模型還是一樣error。後來才發現是jupyter notebook在重新跑的時候，原先舊的模型instance沒有回收導致他們都堆積在GPU RAM裡面。後來查了一些資料，原本想說weakref可以解決這個釋放的問題（因為我原本把instance直接寫在list中，而不是把Class放在裡面），後來發現這個沒辦法直接在訓練完成後送他去垃圾桶，最後還是乖乖尋找釋放RAM的方法，才解決這個問題。
  + 一如既往的，思考要畫哪些圖和整理數據畫出來是個繁瑣的過程，不過至少稍微學會怎麼畫自己想要的圖表了，雖然還有一些訓練的數據來不及整理完整。
  + 這次的任務在Colab上訓練時間比我的電腦快速許多，穩定度也比起前兩次作業高了不少，所以這次程式碼是在本地撰寫，後續 fine-tune和數據蒐集都使用Colab，直到沒有額度為止才換回我自己的電腦。兩者環境如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Running environment | System | CPU | GPU | Python version |
| Local | Windows 11 Version 23H2 (KB5046633) | 11th Gen Intel® Core™ i7-11370H @ 3.30GHz | NVIDIA GeForce RTX 3050 Ti Laptop GPU | CPython 3.11.1 |
| Google Colab | Colab | Colab | T4 GPU | Colab |

※註：本作業部分程式碼使用ChatGPT-4o生成，部分程式碼使用GitHub Copilot協助撰寫，區塊皆於檔案中註解。