DLCV HW4 Report

學號 ：R08945006 學生 ： 葉政樑

Problem1-1: Prototypical Network

Architecture:

在 feature extractor 網路架構方面我使用了 Conv. Block 堆疊而成，網路共由 4 層 Conv. Block 組成，最終輸出 H*W*1600 channels。在每層 Conv. Block 中由(Conv2d + BatchNorm2d + ReLU + MaxPool2d)組成，並且使用高斯分布 weight initialization。

Implementation details:

首先在 implement 時發現比較麻煩的地方是如何產生出多個 episode，或者說多個 task，一開始發現自己在 sample 時有寫錯，因此重複 sample 到一樣類別的 data，所以訓練效果不好，對於一些常看到的 class 很快會 overfit。後續修改則參考助教提供的 test_testcase.py 與網路上的一些 sampler 寫法後，成功修改了這個 bug。

Training episodes: 100000 (1000epoches，每個 epoch 有 100episodes)

Distance function: Euclidean distance (cosine 表現不好)

Optimizer and learning rate: Adam(model.parameters(), lr=0.001)

Learning rate schedule: lr_scheduler.StepLR(optimizer, step_size=100, gamma=0.5)

Data Augmentation: Normalize: mean= [0.485, 0.456, 0.406]，std=[0.229, 0.224, 0.225]

N-way K-shot for meta-train: 5way-1shot-15query

N-way K-shot for meta-test: 5way-1shot-15query

The Acc. on validation set under 5-way 1-shot setting: 49.73 +- 0.85 %

Problem1 參考資料:

1. Jake Snell, Kevin Swersky, Richard S. Zemel. *Prototypical Networks for Few-shot Learning.* arXiv:1703.05175 [cs.LG]: *https://arxiv.org/pdf/1703.05175.pdf*
2. 李宏毅老師 Meta Learning: https://www.youtube.com/watch?v=semSxPP2Yzg&list=PLJV_el3uVTsOK_ZK5L0Iv_EQoL1JefRL4&index=46
3. *Github:* *https://github.com/yinboc/prototypical-network-pytorch*
4. Pytorch Sampler 詳解: https://zhuanlan.zhihu.com/p/82985227
5. Pytorch official: https://pytorch.org/docs/stable/data.html

Problem1-2: using 3 different distance function

同樣使用 5-way 1-shot setting，為節省訓練時間只 training 5000 episodes，結果如下:

Euclidean distance ACC.= 38.52 +- 0.72 %

使用 Euclidean distance 效果很好，在訓練初期即可看到 ACC.穩定上升。無意間發現問題: 是否開平方根號似乎影響不大? 兩種測試後都可以訓練成功，最終以原始公式有開根號的結果為準。

Cosine similarity ACC.= 30.54 +- 0.65 %

使用 Cosine similarity 的效果遜色於 Euclidean distance，於 prototypical network 原始論文中作者亦是得到如此結論，因此實驗結果吻合。

Parametric function ACC. = 28.64 +- 0.63 % (似乎沒有 training 成功)

在 parametric function 方面，1. 我將 support set 和 query set 都經過 feature extractor 後，support set 去算 prototype，而 query set 則再經過一個 MLP 架構(input、output 都為 1600channels)，最後將 prototype 與 MLP output 相減取絕對值後再取 argmin 為 prediction。將 prediction 與 label 做 cross entropy loss。發現問題: 這樣設計似乎只有將 query set 額外通過 MLP，最終似乎還是以相減取絕對值當作 distance function。2. 接著我再嘗試將 support set 與 query set 接在一起直接經過 MLP，最終 MLP output channel = 5 ，分別對應 5-way 的五個類別，最終用 cross entropy loss 去計算，最後發現效果不是很好，不確定是 MLP 架構設計問題或是 loss function 寫錯，抑或是 training episode 不夠。

Problem1-3: meta-train and meta-test under the same 5-way K-shot setting, (K=1, 5, 10)

為節省訓練時間只 training 5000 episodes，除 K-shot 改變以外其餘條件皆相同。

	5-way 1-shot	5-way 5-shot	5-way 10-shot
Accuracy	38.52+-0.72%	38.30+-0.68%	39.23+-0.73%

從上述結果來看發現 1shot 和 5shot 在訓練 5000episodes，Acc 相差不大，而 10shot 則是略高於 1shot 以及 5shot，此結果符合我的推論: support set 較多算出的 prototype 較準確，則 ACC. 表現較好。

Problem2-1: Data Hallucination for Few-shot Learning

Architecture:

在 feature extractor 架構方面我沿用了 problem1 的 Conv. Block 堆疊而成網路，另外參考了"Low-Shot Learning from Imaginary Data"這篇 paper 的架構，在這篇 paper 中的架構多了 hallucinator G，G 的用途在於產生出能幫助 classification task 的 imaginary data，G 為一個 MLP 架構，以 linear + ReLU 組成。實做的流程大致上分為 1. 產生出每一個 episode data。 2. 對於每一個 N-way 隨機從 k-shot 中 sample 一張 data。 3. 從 gaussian distribution 中 sample noise。 4. 將 2. 3. 一起丟進 hallucinator G。 5. 將原 N-way K-shot 的 data 與 hallucinator G 產生的 augmentation data 一起丟進 prototypical network 做 meta learning classification task。

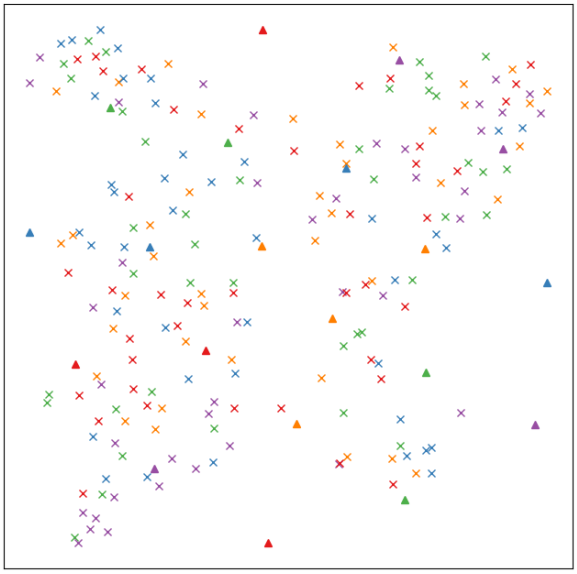
Implementation details:

在實作上多了 sample noise from gaussian distribution 與 sample data from K-shot 這兩個步驟，並且參考原始 paper 中 G 使用簡單的 MLP 架構，最終再接上 Problem1 的 prototypical

network 即可達到 data hallucination end to end training 的結果。
Training episodes: 100000 (1000epoches，每個 epoch 有 100episodes)
Distance function: Euclidean distance
Optimizer and learning rate: Adam(model.parameters(), lr=0.001)
Learning rate schedule: lr_scheduler.StepLR(optimizer, step_size=100, gamma=0.5)
Data Augmentation: Normalize: mean= [0.485, 0.456, 0.406]，std=[0.229, 0.224, 0.225]
N-way K-shot M-augmentation for meta-train: 5way-1shot-100M-15query
N-way K-shot M-augmentation for meta-test: 5way-1shot-100M-15query
The Acc. on validation set under 5-way 1-shot setting: 49.01 +- 0.88%

Problem2-2: t-SNE

右圖為 t-SNE 視覺化結果。
X 代表 200 real data， 三角形代表 20 hallucinated data，其中又以顏色分成 5 個 Class (5way)。
特別注意到我的結果與助教在 ppt 上的 example 表現出不同結果，並沒有將 real data 與 hallucinated data 分得很開，我猜想是否是 hallucinator 認為這樣產生出的 data 能夠與 real data 較為相近，並且對於 classification task 較有幫助。



Problem2-3: meta-train, meta-test under the same 5-way 1-shot M-augmentation (M=10,50,100)

為節省訓練時間只 training 10000 episodes。(M=100 則使用 problem2-1 結果)

	5-way 1-shot M-10	5-way 1-shot M-50	5-way 1-shot M-100
Accuracy	43.53 +- 0.81 %	46.18 +- 0.86 %	49.01 +- 0.88 %

從上述結果來看發現使用不同的 M 對 Acc.是有一定影響的，而 M = 100 是我訓練出最好的結果，藉由比較不同 M 的過程中我發現單純從不同 M 比較結果上有一定難度，有可能 M 越大需要的 episodes 較多才能達到更好的結果。因此都用同樣的 episodes 有時 M 越大不一定表現越好，有可能是還在 underfitting。

Problem2-4: Discuss what you've learned from implementing the data hallucination model.

本次作業是我第一次接觸 meta learning 的實做與進階使用 data hallucination for few-shot learning。在一開始我不太熟悉 meta learning 的 task (episode)應該如何產生，我發現這是一個比較麻煩的地方，後續藉由熟悉助教給予的 code 之後成功產生出 N-way K-shot episode 來

訓練。而在 data hallucination 時又碰上一次不知道該怎麼做的問題，後來參考了 Paper: ”Low-Shot Learning from Imaginary Data” 與助教 ppt 的講解，成功理解應該從每個 N-way 中 K-shot 裡面隨機 sample，然後再把從 gaussian distribution 裡 sample 的 noise 接起來再一起丟進去 G，最後才把 K+M data 拿去算 prototype。這邊回頭來看算是很清楚的流程，但是在我第一次看到時有點被搞混究竟甚麼時候應該用 data hallucination，q-query 要用嗎?這類型的問題一直搞混。

第二則是在檢驗使用 data hallucination 是否有幫助時，我發現有時候使用反而效果更差，一開始以為是自己有寫錯 model，或是 sample 時有問題，又或是 hallucinator 架構不對，其中問題一直不知道在哪裡，前前後後花了不少時間修改，最後發現有可能是 training episode 不夠多，基於試驗我都是 training 10000 episode，發現 M 比較大時根本還沒 training 好，我猜想可能是 hallucinator 也需要一些 training 時間去學會如何產生出對 classification 有幫助的 data。因此後續我又回去參考原始 paper，發現他們其實做了很多分析，不單單是像我這樣 naïve 的比較 Acc.而已。

最後我認為 few-shot learning 這部分算是以前從沒碰過的領域，我學習到很多新東西，也知道了更多更前沿的研究，透過參考一些新的 paper 會有大開眼界的感覺。

Problem2 參考資料:

1. Yu-Xiong Wang[1,2], Ross Girshick[1], Martial Hebert[2], Bharath Hariharan[1,3]. *Low-Shot Learning from Imaginary Data.* CVPR2018
   https://openaccess.thecvf.com/content_cvpr_2018/papers/Wang_Low-Shot_Learning_From_CVPR_2018_paper.pdf
2. Pytorch torch.utils:
   https://pytorch.org/docs/stable/data.html
3. Sklearn t-SNE:
   https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html

Problem3:

在第三題我試著 implement ”Diversity Transfer Network for Few-Shot Learning. AAAI 2020” 這篇 paper 的內容，然而在 implement 和 training 上遇到許多問題，直到作業截止前仍未有結果出現，目前正在重新調整架構與修改程式碼。