HW4
陳芃彣 r08942085

Problem 1-1

Describe the architecture & implementation details of your model. (Include but not limited to the number of training episodes, distance function, learning rate schedule, data augmentation, optimizer, and N-way K-shot setting for meta-train and meta-test phase)

Architecture:
我的模型架構直接使用助教提供之 Conv-4，並沒有做任何修改，也並未加 MLP，因此模型架構就是由四個 convolution block 搭建而成，convolution block 由一層 CNN，一層 BatchNormalize，一層 ReLU 以及一層 MaxPooling 構成，就如同助教給的 convolution block。

Implementation details:
meta-train 時我使用助教的提示做 10 way 1 shot，query 共 7 組，所以總共有 70 張 query，並生成 2400 個 training episodes；meta-test 如同題目規定，使用助教提供的 5 way 1 shot 的 val_testcase.csv。

Distance function: Euclidean metric
Learning rate schedule: 每 20 個 epoch，將 lr 乘以 0.5，初始 lr 為 1e-3。
Date augmentation: 無
Optimizer: Adam

Accuracy:

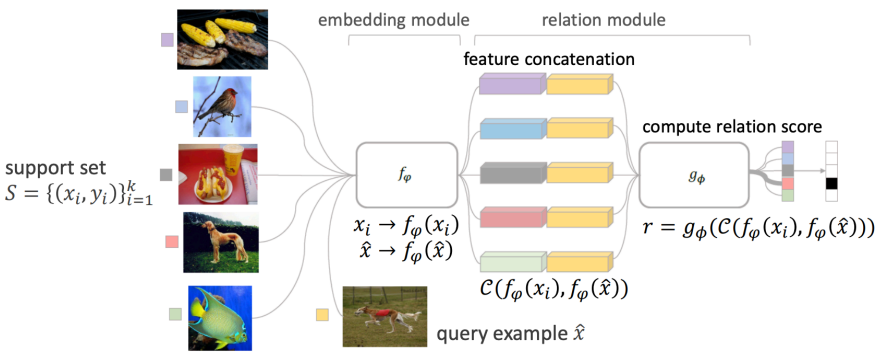
(about epoch 10)

Problem 1-2

When meta-train and meta-test under the same 5-way 1-shot setting, please report and discuss the accuracy of the prototypical network using 3 different distance function (i.e., Euclidean distance, cosine similarity and parametric function). You should also describe how you design your parametric function.

	Euclidean	Cosine	parametric (Relation Net)
Accuracy (10 epoch)	44.20 +- 0.83%	36.13 +- 0.75%	43.32 +- 0.80%

由 Accuracy 可以看出來在同樣的 10 epoch 下，Euclidean metric 的成績最高、收斂最快，cosine similarity 比較低分，可能原因為 Euclidean distance 是計算兩點之間的距離，consine similarity 則是計算兩得 vector 之間的角度 (沒有考慮 magnitude) 。 我使用 Relation Network 作為我的 parametric function，模型架構如下圖所示（圖擷取自老師上課投影片 W12 p.37）。



由上圖可見，support and query set 經過 feature extractor 後，會將每個 support feature 與 query feature concat 在一起，再丟入 compute relation net 計算兩者之間的相似度成績，而我的 compute relation net 是由兩層 CNN 與三層 fc + sigmoid 搭建，詳細可見下方截圖。

```python
class Parametric(nn.Module):
    def __init__(self, n_way, in_channels=3, hid_channels=64, out_channels=64):
        super().__init__()
        self.encoder = nn.Sequential(
            conv_block(in_channels, hid_channels),
            conv_block(hid_channels, hid_channels),
            conv_block(hid_channels, hid_channels),
            conv_block(hid_channels, out_channels),
        )
        self.relation = nn.Sequential(
            conv_block(hid_channels*2, hid_channels),
            conv_block(hid_channels, hid_channels),
        )
        self.fc = nn.Sequential(
            nn.Linear(320, 128),
            nn.Linear(128, 32),
            nn.Linear(32, n_way),
            nn.Sigmoid(),
        )

    def forward(self, support, query):
        support = self.encoder(support)
        query = self.encoder(query)

        query = torch.unsqueeze(query, dim=1)
        query = query.repeat(1,5,1,1,1)
        query = query.reshape(75*5,64,5,5)

        support = support.repeat(75,1,1,1)

        dis = torch.cat((support, query), dim=1) # 375*64*5*5
        dis = self.relation(dis) # 375*64*1*1
        dis = dis.reshape(75,5,64,1,1)
        dis = dis.view(dis.size(0), -1)
        dis = self.fc(dis)
        return dis
```

feature extractor

compute relation

使用 relation net 同時訓練 feature extractor 與預測 similarity distance 成績有比使用 cosine similarity 高一些，然而因為 similarity distance 計算變成由 neural net 計算，整體收斂速度會變得比較慢，而且也未必會比 prototypical + euclidean 好。

Problem 1-3

When meta-train and meta-test under the same 5-way K-shot setting, please report and compare the accuracy with different shots. (K=1, 5, 10)

	5 way 1 shot	5 way 5 shot	5 way 10 shot
Accurace	44.20 +- 0.83% (epoch 10)	61.90 +- 0.33% (epoch 8)	66.89 +- 0.54% (epoch 3)

可以很明顯的發現 shot 數變大，有助於後續 query 的分類，除了收斂變快，準確度也上升很多，因為 shot 數變大等同於資料量變多，若 shot 數變得無限多其實也就變成一般的監督是學習。

Problem2-1

Describe the architecture & implementation details of your model. (Include but not limited to the number of training episodes, distance function, learning rate schedule, data augmentation, optimizer, and N-way K-shot M-augmentation setting for meta-train and meta-test phase)

Architecture:
Feature Extructor 與上一題相同，Hallucinator 我使用三層 fc，hidden size 皆為 1600，並且每層之間有加 ReLU 與 Dropout，最後 MLP 僅用了一層 fc，hidden size 一樣是 1600。

其中，我是用 torch.randn sample 出 Hallucinator 的 noise input，並且使用相加的方式把 noise 加進原始 data 的 feature，再經過 Hallucinator 生成假 feature，然後將真假 feature concat 後再丟入 MLP。

```python
class Convnet(nn.Module):
    def __init__(self, in_channels=3, hid_channels=64, out_channels=64):
        super().__init__()
        self.encoder = nn.Sequential(
            conv_block(in_channels, hid_channels),
            conv_block(hid_channels, hid_channels),
            conv_block(hid_channels, hid_channels),
            conv_block(hid_channels, out_channels),
        )
        self.hallucination = nn.Sequential(
            nn.Linear(1600, 1600),
            nn.ReLU(),
            nn.Dropout(),
            nn.Linear(1600, 1600),
            nn.ReLU(),
            nn.Dropout(),
            nn.Linear(1600, 1600),
            nn.ReLU(),
            nn.Dropout(),
        )
        self.MLP = nn.Sequential(
            nn.Linear(1600, 1600),
            nn.ReLU(),
        )
```

```python
    def forward(self, x, M=1, noise=None, query=False, halluci=False):
        x = self.encoder(x)
        ori_data = x.view(x.size(0), -1) # (10,1600)
        if halluci:
            repeat_data = ori_data
            data = repeat_data + noise
            aug_data = self.hallucination(data)
            output = self.MLP(aug_data)
            return output
        if not query:
            repeat_data = ori_data.repeat(M,1)
            data = repeat_data + noise
            aug_data = self.hallucination(data)
            output = torch.cat((ori_data, aug_data), dim=0)
            output = self.MLP(output)
        else:
            output = self.MLP(ori_data)
        return output
```

Support goes here

Query goes here

Implement Details:
Training episodes: 2400
Distance function: Euclidean distance
Learning Rate Schedule: 每 20 個 epoch lr 乘以 0.5，初始 lr 為 1e-3
Data augmentation: 無
Optimizer: Adam
Meta-train: 10 way 1shot, M=200

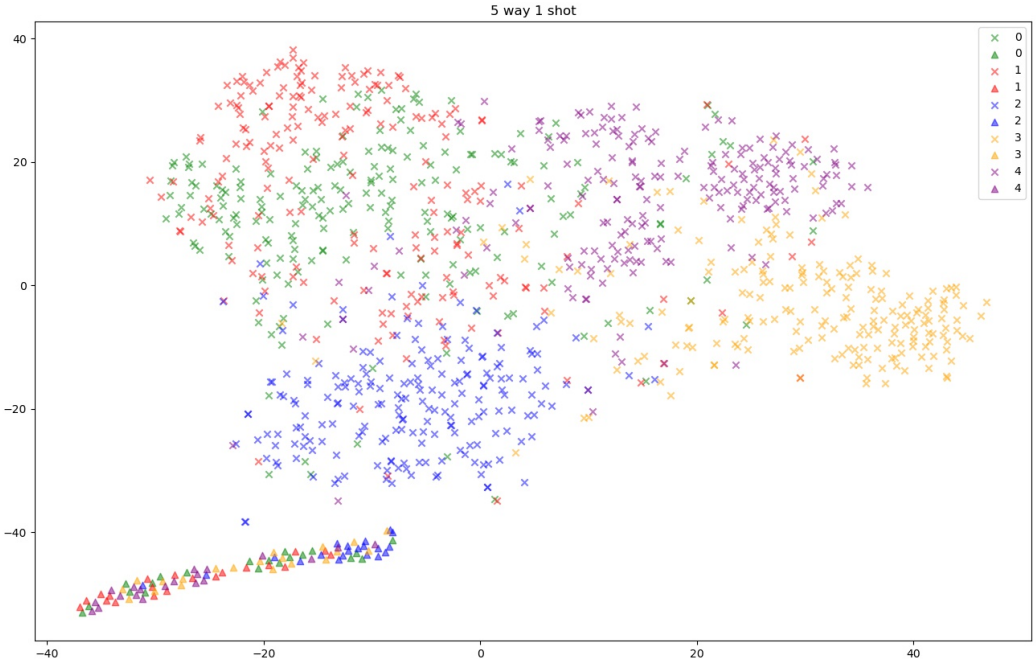
Meta-test: 5 way 1 shot, M=200
Accuracy:

(epoch 4)

```
Accuracy: 47.09 +- 0.89 %
```

Problem 2-2

To analyze the quality of your hallucinated data, please visualize the real and hallucinated (training) data in the latent space (the space where you calculate the prototypes of each class, e.g., the output of the MLP layer) by mapping the features to 2D space (with t-SNE). Briefly explain your result of t-SNE visualization. Example for fig2-3.png is shown below.



透過 tone 圖可以看到 real data 中，五個 class 有明顯被區分出來，但是 fake data 完全沒有區分效果，可見也許單用 fc 做 hallucinator 可能難以生出與 real data 非常相似的 fake data。

Problem 2-3

When meta-train and meta-test under the same 5-way 1-shot M-augmentation setting, please report and compare the accuracy with different number of hallucinated data. (M=10, 50, 100)

	M=10	M=50	M=100
Accuracy	41.95 +- 0.82% (epoch 5)	42.01 +- 0.01% (epoch 5)	43.39 +- 0.31% (epoch 5)

M=10 與 M=50 比較，M=50 準確率有稍微上升，可以理解為 M 越多，確實有做到些微的 data augmentation，但是模型收斂得越慢，可能原因為 M 越多，假資料不像真資料的張數也就越多，因此訓練比較慢。

Problem 2-4

Discuss what you've observed and learned from implementing the data hallucination model.

訓練 Data hallucination 時，M 影響很大，當 M = 200 時，在 epoch 4 時過了 baseline，相較於單純的 prototypical network，Data hallucination 收斂的比較快， 然而透過 tsne 圖可以看出來 hallucinator 生成的假資料是否與真資料相近這點尚有疑慮，因此將 hallucinator 從單純僅用 fc 更改成其他架構，例如 GAN，可能會有更好的 data augmentation 效果。

Problem 3-1

Describe the architecture & implementation details of your model. (Include but not limited to the number of training episodes, distance function, learning rate schedule, data augmentation, optimizer, and N-way K-shot M-augmentation setting for meta-train and meta-test phase)

Problem 3-2

To analyze the quality of your hallucinated data, please visualize the real and hallucinated (training) data in the latent space (the space where you calculate the prototypes of each class, e.g., the output of the MLP layer) by mapping the features to 2D space (with t-SNE).

Problem 3-3

Try to explain why the improved model performs better than the one in Problem 2 (e.g., discuss the difference between your visualization of latent space in Problem 2 and Problem 3).