DLCV HW4

學號 r08945050
姓名 蔡宇晴

Report (50%)

1. (20%) 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)
   Please report the accuracy on validation set under 5-way 1-shot setting (during inference).
   - **The accuracy should be the same as your model performance in P.8.**
   - **TAs will run your code to verify the performance**

Model

```
Convnet(
  (encoder): Sequential(
    (0): Sequential(
      (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU()
      (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
    (1): Sequential(
      (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU()
      (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
    (2): Sequential(
      (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU()
      (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
    (3): Sequential(
      (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU()
      (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
  )
)
```

Hyperparameters:

```
lr = 0.001
epoch = 300
```

Optimizer:

```
optimizer = torch.optim.Adam(model.parameters(), lr=lr)
```

Performance :

```
Accuracy: 46.95 +- 0.85 %
```

Distance function:

1. Euclidean distance

```
def euclidean_distance(self, query, mean):
    N_query = query.shape[0]
    N_way = mean.shape[0]
    query = query.unsqueeze(1).expand(N_query, N_way, -1)
    mean = mean.unsqueeze(0).expand(N_query, N_way, -1)
    logits = -((query - mean) ** 2).sum(dim=2)
    return logits
```

2. Cosine Similarity

```
def cosine_similarity(self, query, mean):
    N_query = query.shape[0]
    N_way = mean.shape[0]
    query = query.unsqueeze(1).expand(N_query, N_way, -1)
    mean = mean.unsqueeze(0).expand(N_query, N_way, -1)
    dot = (query * mean).sum(dim=2)
    norm_query = (query * query).sum(dim=2) ** 0.5
    norm_mean = (mean * mean).sum(dim=2) ** 0.5
    return dot / (norm_mean * norm_query)
```

3. Parametric function (learnable)

```python
class Dist_Net(nn.Module):
    def __init__(self, N_way):
        super(Dist_Net, self).__init__()
        self.fc = nn.Sequential(
            nn.Linear(2 * N_way * 1600, 1024),
            nn.ReLU(True),
            nn.Linear(1024, 512),
            nn.ReLU(True),
            nn.Linear(512, 64),
            nn.ReLU(True),
            nn.Linear(64, N_way),
        )
    def forward(self, x):
        x = self.fc(x)
        return x
```

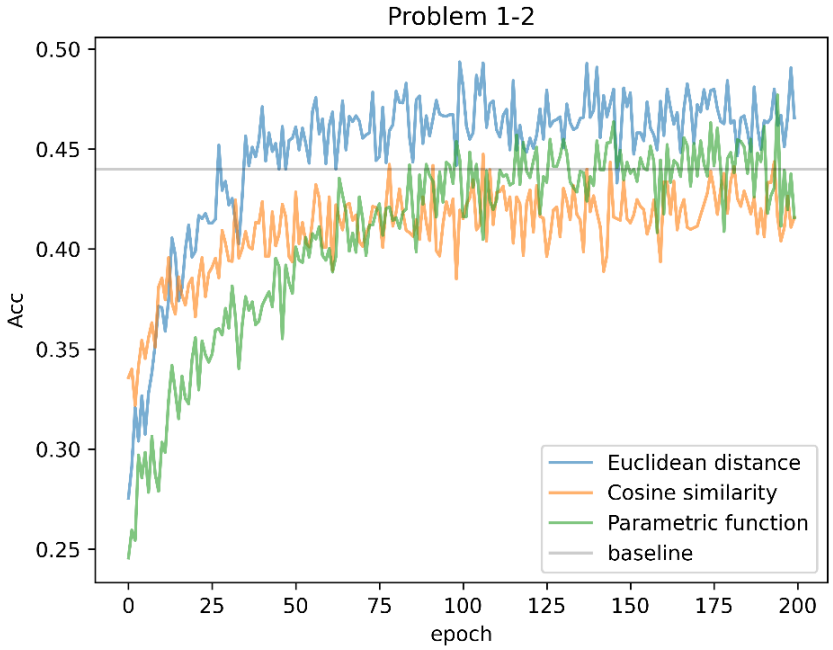
Discussion:

Prototypical Network 的核心思想在於，若透過某個 feature extractor 將 data 轉換至另一個 embedded space，則相同類別的資料的歐式距離應該越接近 or 相似度應較高，不同資料則相反。然而不同類別的資料在 embedded space 可能會有不同的離散程度，因此不同類別的 proto 來自於相對應類別的平均。
這個架構的缺點為，若 shot 的數量少，則相對應代表 proto 不足以代表整個 class 的特性，此時等同於在拿個一個錯誤的 label 在訓練 network。

2. (20%) 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.

   我選擇了多層 Linear 當作 **parametric function(上圖)**，希望可以透過 FC 使得

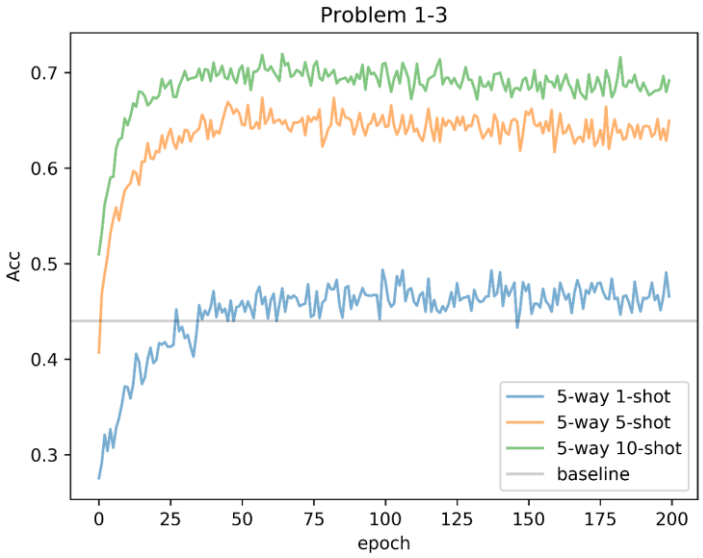
embedded space 的向量可以自動被判別成與 proto 一樣的類別，而不是單純的透過距離公式 or 相似度公式來決定 embedded space 的向量屬於哪個類別。因為相似度公式與距離公式照理說並非適用於任何 space 的向量，而每個特定的 embedded space 或許都有對應最合適的計算相似度的方式。但實際結果仍以 Euclidean distance 的訓練結果最好。結果如下圖。



結論解釋:

我認為在訓練 FC 時，可以將 Convnet 先進行 pre-train(**上圖的 parametric function 是與 Convnet 同時訓練的，而不是使用 pre-trained 的 Convnet**)，例如使用歐式距離 train 過的 Convnet，接著再 fine-tune 後面的 FC 層。而一起訓練可能導致 Convnet 和 FC 層的 loss 都過大而無法收斂。使用預訓練後的正確率約會收斂在 0.46，比起沒有 pretrained 的 model ，正確率似乎有些微提升。

3. (10%) 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)



由上圖可以看出，當 shot 越高時，acc 會隨之上升，其原因是由於少量的圖片(shot)可能無法代表一個好的 proto，而隨著圖片增加，proto 也更具代表性。

Model Performance (20%)

- The mean accuracy of your model under 5-way 1-shot setting (during inference) (Problem 1-1) should be above the baseline score.

  Ans:My Acc :46.95%

Problem 2: Self-Supervised Pre-training for Image Classification (50%)

1. (10%) Describe the implementation details of your SSL method for pre-training the ResNet50 backbone. (Include but not limited to the name of the SSL method you used, data augmentation for SSL, learning rate schedule, optimizer, and batch size setting for this pre-training phase)
   - 使用 BYOL
   - data augmentation(如下圖)

```
DEFAULT_AUG = torch.nn.Sequential(
    RandomApply(
        T.ColorJitter(0.8, 0.8, 0.8, 0.2),
        p = 0.3
    ),
    T.RandomGrayscale(p=0.2),
    T.RandomHorizontalFlip(),
    RandomApply(
        T.GaussianBlur((3, 3), (1.0, 2.0)),
        p = 0.2
    ),
    T.RandomResizedCrop((image_size, image_size)),
    T.Normalize(
        mean=torch.tensor([0.485, 0.456, 0.406]),
        std=torch.tensor([0.229, 0.224, 0.225])),
)
```

- lr = 3e-4
- Optimizer: Adam
- Batch Size:64

2. (10%) Following Problem 2-1, please conduct the Image classification on Office-Home dataset as the downstream task for your SSL method. Also, please complete the following Table, which contains different image classification setting, and compare the results.

Setting	Pre-training (Mini-ImageNet)	Fine-tuning (Office-Home dataset)	Classification accuracy on valid set (Office-Home dataset)
A	-	Train full model (backbone + classifier)	*23%*
B	w/ label (TAs have provided this backbone)	Train full model (backbone + classifier)	*35%*
C	w/o label (Your SSL pre-trained backbone)	Train full model (backbone + classifier)	*46%*
D	w/ label (TAs have provided this backbone)	Fix the backbone. Train classifier only	*36%*
E	w/o label (Your SSL pre-trained backbone)	Fix the backbone. Train classifier only	*44%*

3. (10%) Discuss or analyze the results in Problem 2-2

5 種正確率如下圖，正確率由大到小依序為:C->E->D->B->A。

A. 沒有 pre-training 的前提下，模型的 resnet50 的狀態是初始的，此時模型並未具有任何抽取特徵的能力，因此在此前提下進行訓練的結果是最糟糕的。
B. 在提供的參數下進行訓練，由於 resnet50 已在一定程度下的訓練，模型具有基本的特徵抽取能力，在此前提下，訓練結果優於 A，但是結果與 D 並沒有相差太多。
C. 透過 BYOL 的長時間訓練，最後在 400 epoch 時，loss 下降至最低，此時的 resnet50 比起 B 的具有更強的特徵抽取能力，接著透過整個架構的 fine-tune，會使得正確率最高。
D. B 和 D 的情況類似，但是 D 卻有訓練整個架構，已以往的經驗結果應該要

比 B 還要好，但實際情況其實結果相近。

E. E 和 C 情況相似，而且結果較符合預期，由於 E 只有訓練 classifier 並 fix 住 backbone，這會使得特徵抽取無法再經由 office home 的資料進行調整，但其實結果只有差距 2~3 個百分點。

此外，在 ABD 中，我有參考相關資料，設定 fine-tune 時不同 layer 的 lr 數值，如下:

```
params = [
    {'params':net.conv1.parameters(),'lr':lr/10},
    {'params':net.bn1.parameters(),'lr':lr/10},
    {'params':net.layer1.parameters(),'lr':lr/8},
    {'params':net.layer2.parameters(),'lr':lr/6},
    {'params':net.layer3.parameters(),'lr':lr/4},
    {'params':net.layer4.parameters(),'lr':lr/2},
    {'params':net.fc.parameters(),'lr':lr/10},
]
```

若未經過此設定，fine-tune 訓練過程似乎 loss 收斂的很慢甚至會收斂在較差的 Acc。

Setting	Pre-training (Mini-ImageNet)	Fine-tuning (Office-Home dataset)	Classification accuracy on valid set (Office-Home dataset)
A	-	Train full model (backbone + classifier)	*23%*
B	w/ label (TAs have provided this backbone)	Train full model (backbone + classifier)	*35%*
C	w/o label (Your SSL pre-trained backbone)	Train full model (backbone + classifier)	*46%*
D	w/ label (TAs have provided this backbone)	Fix the backbone. Train classifier only	*36%*
E	w/o label (Your SSL pre-trained backbone)	Fix the backbone. Train classifier only	*44%*

Model Performance (20%)

- Classification accuracy under setting C in Problem 2-2 should be above the baseline score to get points

  My ACC : 46.3%