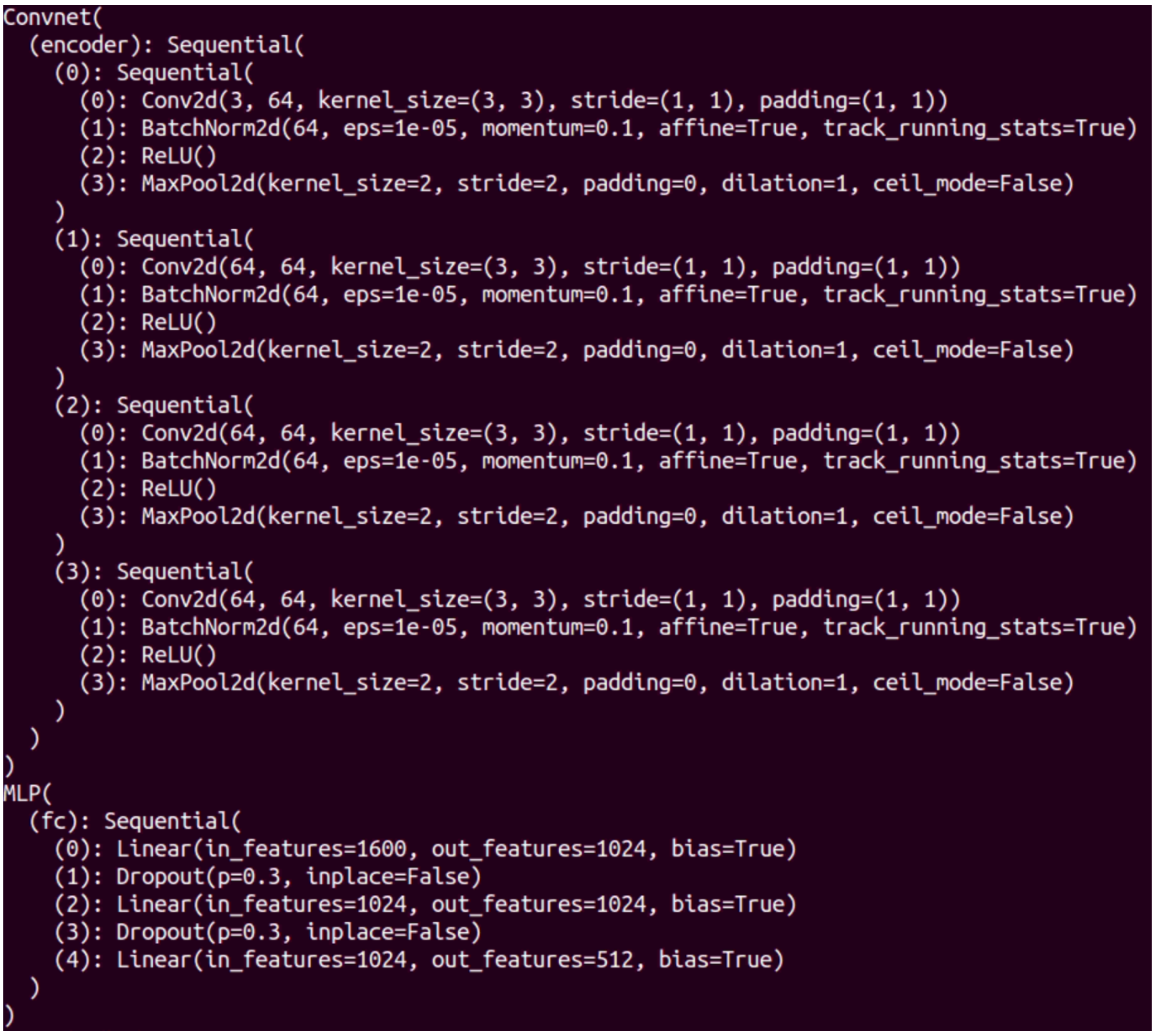
DLCV Homework 4

r09944003 網媒所碩一 陳竣宇

Collaborators: r09922025 張凱程

Problem 1: Prototypical Network

1. - Network architecture



```
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)
    )
  )
)
MLP(
  (fc): Sequential(
    (0): Linear(in_features=1600, out_features=1024, bias=True)
    (1): Dropout(p=0.3, inplace=False)
    (2): Linear(in_features=1024, out_features=1024, bias=True)
    (3): Dropout(p=0.3, inplace=False)
    (4): Linear(in_features=1024, out_features=512, bias=True)
  )
)
```

   - Implementation details
     - Training epochs: 150
     - Training episodes: 600
     - Distance function: euclidean distance
     - Optimizer: SGD (learning rate = 0.001)
     - No data augmentation

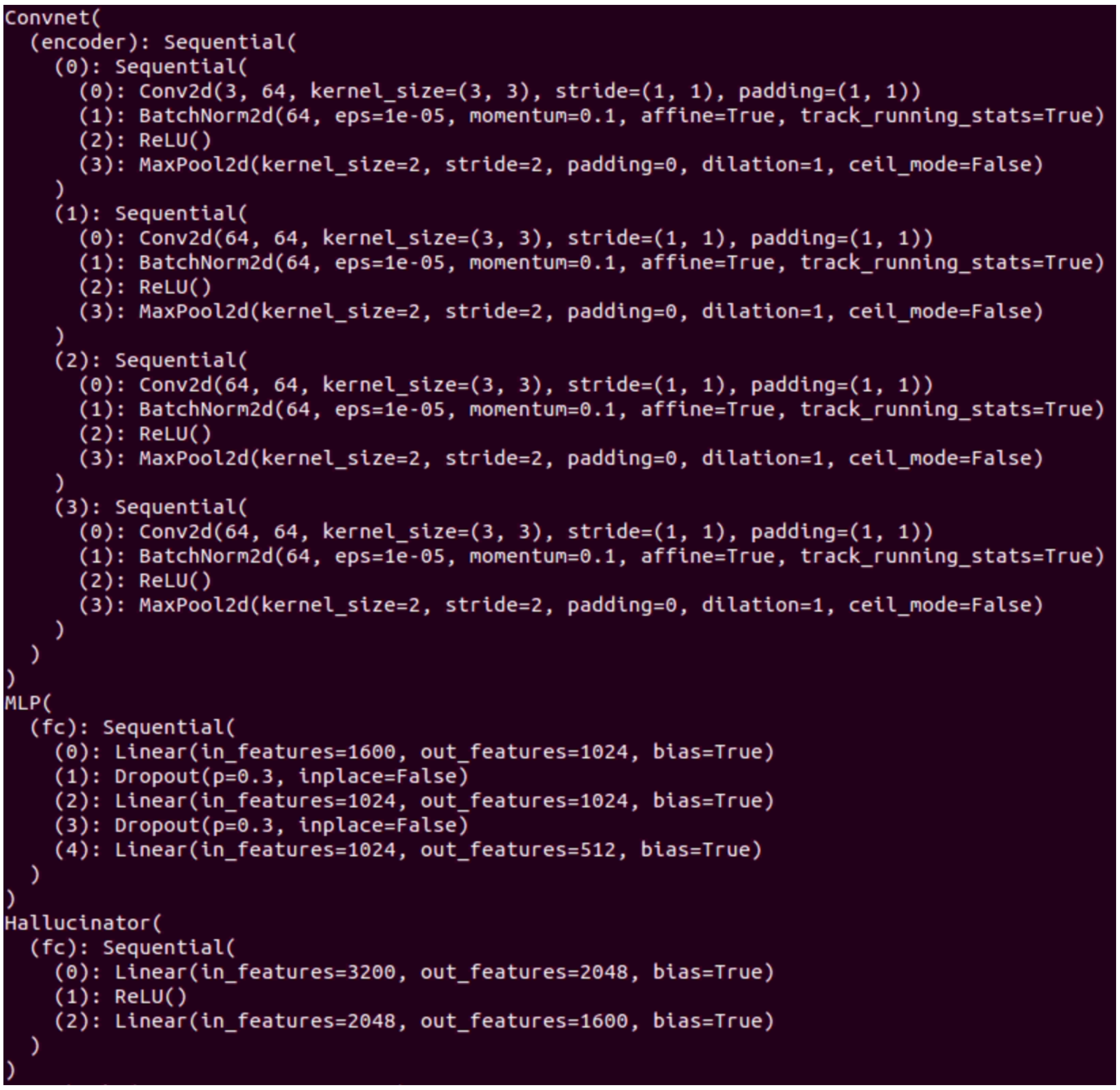
        - 10-way 1-shot for meta-training
        - 5-way 1-shot for meta-testing
    - The accuracy on validation set is **0.4561**

2. For this problem, I train 50 epochs for each experiment of different distance function.
    - euclidean distance: **0.4301**
    - cosine similarity: **0.3854**
    - parametric function: **0.4367**
    - Discussion
        - I build a fully-connected layer **(Linear(512, 512))** to measure the distance between the two features. After feeding the query feature into this model and obtrain the output, I just do a matrix multiplication between this output and the transpose of the prototype feature. The final result is the distance(score) between the two features.
        - And obviously, using the parametric function as the distance provide the best accuracy to our task. I guess that the learnable mechanism leads to this final result.

3. For this problem, I train 50 epochs for each experiment of different K.
    - K = 1: **0.4301**
    - K = 5: **0.44**
    - K = 10: **0.4202**
    - Discussion
        - According to the experimental result, increasing the number of shots appropriately can add the diversity of the support set and make the model more robust.

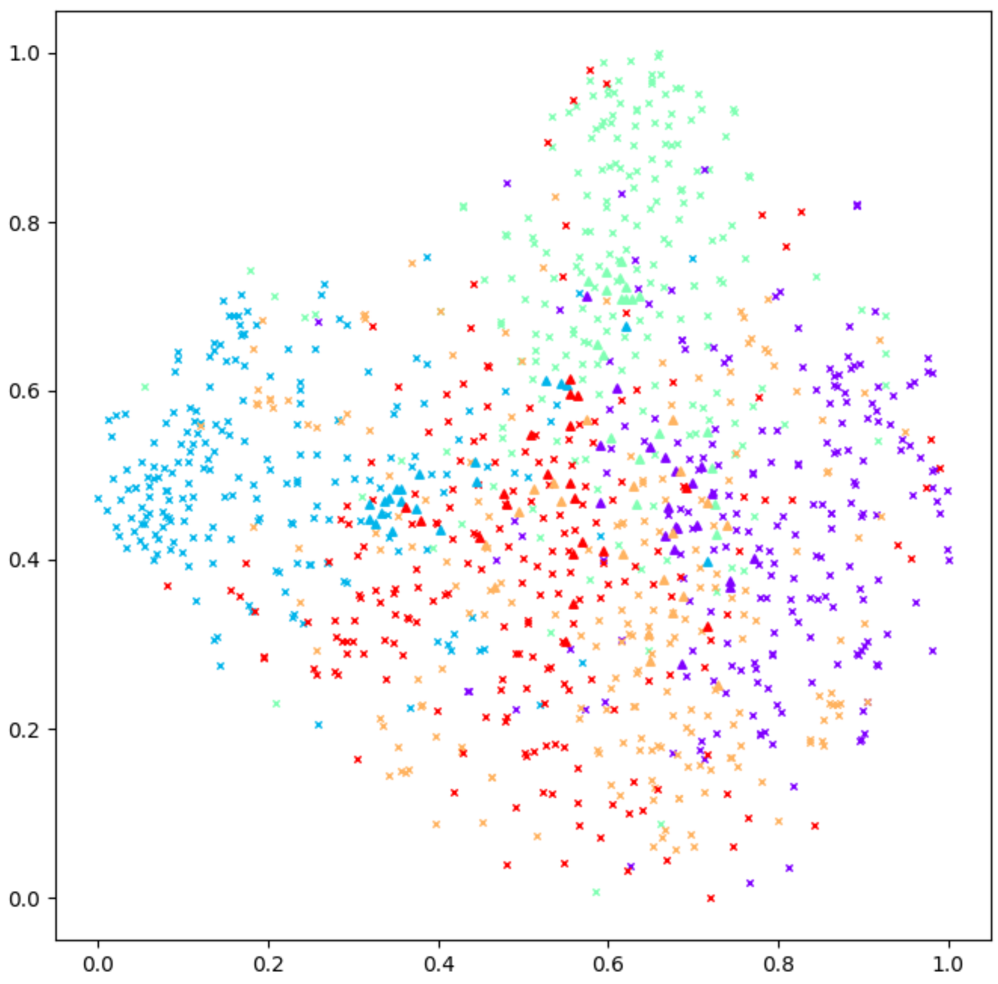
Problem 2: Data Hallucination for Few-shot Learning

1. - Network architecture



```
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)
    )
  )
)
MLP(
  (fc): Sequential(
    (0): Linear(in_features=1600, out_features=1024, bias=True)
    (1): Dropout(p=0.3, inplace=False)
    (2): Linear(in_features=1024, out_features=1024, bias=True)
    (3): Dropout(p=0.3, inplace=False)
    (4): Linear(in_features=1024, out_features=512, bias=True)
  )
)
Hallucinator(
  (fc): Sequential(
    (0): Linear(in_features=3200, out_features=2048, bias=True)
    (1): ReLU()
    (2): Linear(in_features=2048, out_features=1600, bias=True)
  )
)
```

- Implementation details
  - Training epochs: 150
  - Training episodes: 600
  - Distance function: euclidean distance
  - Optimizer: SGD (learning rate = 0.001)
  - 10-way 1-shot 10-augmentation for meta-training
  - 5-way 1-shot 10-augmentation for meta-testing
- The accuracy on validation set is **0.4701**



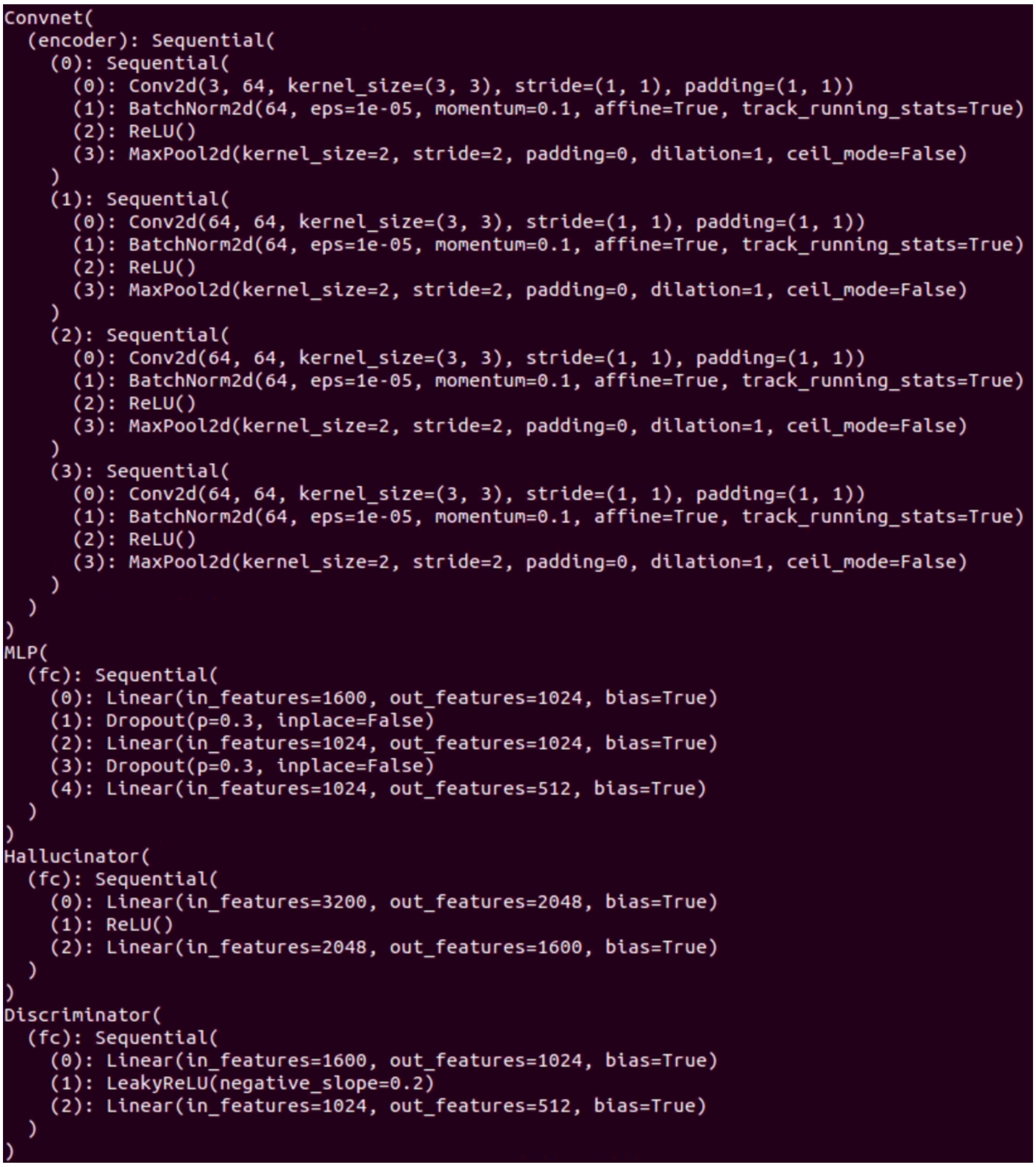
2.

3. For this problem, I train 50 epochs for each experiment of different M.
    - M = 10: **0.4495**
    - M = 50: **0.454**
    - M = 100: **0.4532**
    - Discussion
        - Because we only have 1 shot for each class, hallucinate some fake data with different noises may reduce the bias of the calculated mean and the real mean in the latent space.

4. Discussion
    - 在剛開始做這題時我摸索了好一陣子關於如何hallucinate data以及如何控制input, output的維度等實作細節，後來是透過請教同學才成功train起來。
    - 除了學到如何用hallucinator來做data augmentation之外，我認為像是上一題透過調整M來觀察其對於performance的影響這樣的實驗也能很好的幫助我們體會與理解這些設計想要表達的意涵。

Problem 3: Improved Data Hallucination Model

1. - Network architecture

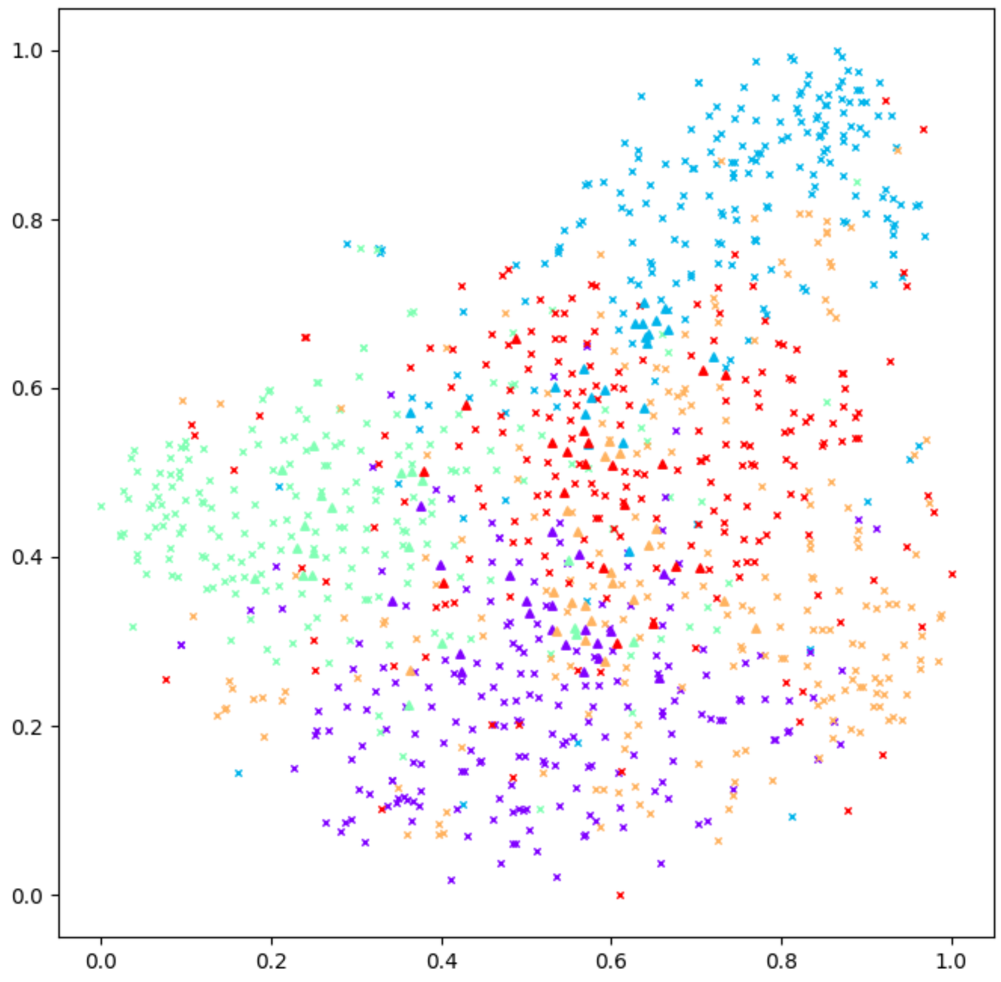


```
Convnet(
  (encoder): Sequential(
    (0): Sequential(
      (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU()
      (3): MaxPool2d(kernel_size=2, stride=2, padding=0, 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)
    )
  )
)
MLP(
  (fc): Sequential(
    (0): Linear(in_features=1600, out_features=1024, bias=True)
    (1): Dropout(p=0.3, inplace=False)
    (2): Linear(in_features=1024, out_features=1024, bias=True)
    (3): Dropout(p=0.3, inplace=False)
    (4): Linear(in_features=1024, out_features=512, bias=True)
  )
)
Hallucinator(
  (fc): Sequential(
    (0): Linear(in_features=3200, out_features=2048, bias=True)
    (1): ReLU()
    (2): Linear(in_features=2048, out_features=1600, bias=True)
  )
)
Discriminator(
  (fc): Sequential(
    (0): Linear(in_features=1600, out_features=1024, bias=True)
    (1): LeakyReLU(negative_slope=0.2)
    (2): Linear(in_features=1024, out_features=512, bias=True)
  )
)
```

   - Implementation details
     - Training epochs: 150
     - Training episodes: 600
     - Distance function: euclidean distance
     - Optimizer: Both generator and discriminator are SGD (learning rate = 0.001, 0.0005)
     - 10-way 1-shot 10-augmentation for meta-training
     - 5-way 1-shot 10-augmentation for meta-testing

- The accuracy on validation set is **0.4811**

2.



3. Discussion
    - 這個task我使用助教在作業說明所提供的第一篇reference來實作，也就是今年在 CVPR 發表的 **Adversarial Feature Hallucination Networks for Few-Shot Learning**這篇論文。
    - 這篇論文提出一個類似WGAN的架構，透過多build一個discriminator來增加生成影像的classification ability和diversity。透過這個設計可以讓我們用hallucinate出來的image去訓練的model更加具有應付不同類型input的能力，也確實讓我在performance上有所提升。
    - 從tSNE的結果來分析，也可以發現相對於第二題的結果，clustering的結果更明顯了一些，生成的data也基本上和原本的input落在相同的分佈上。