HW4 Report

學號: r08942087 姓名: 吳彬睿

Problem 1: Prototypical Network (50%)

1. (11%) Describe the architecture & implementation details of your model.

- Implementation details:
  - Training parameters:
    - Training episodes: 100 epochs * 300 batches = 30000 episodes.
    - Learning rate: 1e-4
    - Optimizer: Adam optimizer
  - Data augmentation:
    - Random crop(scale=(0.5,1.0))
    - Random horizontal flip()
    - Random apply p=0.8, Color Jitter(0.8, 0.8, 0.8, 0.2)
  - Model:

```
Conv4(
  (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)
    )
  )
  (fc): MLP(
    (fc): Sequential(
      (0): Linear(in_features=1600, out_features=512, bias=True)
      (1): Dropout(p=0.5, inplace=False)
      (2): ReLU(inplace=True)
      (3): Linear(in_features=512, out_features=64, bias=True)
    )
  )
```

  - Optimizer:
    - Optimizer: Adam optimizer(lr=1e-4)
  - Prototype learning:
    - Distance function: Euclidean, Cosine, Parametric function.

- Query: 15 query/batch
- Meta-Train: 30-way 1-shot
- Meta-Valid: 5-way 1-shot

➢ Performance:

Backbone	Meta-Train	Meta-Valid	Episodes	Distance F	Valid acc (%)
Conv4	30-way 1-shot	5-way 1-shot	30000	Parametric F	45.97 ± 0.92

➢ My final selected settings:

Meta-Train: 30-way 1-shot

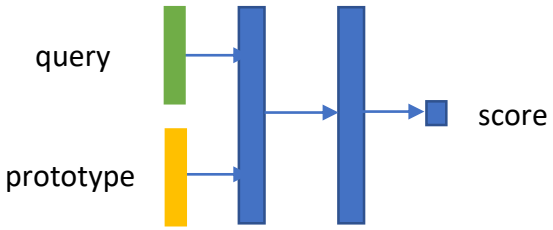
Meta-Valid: 5-way 1-shot

Distance function: Parametric function.

- Validation set accuracy: 45.97 ± 0.92 %

2. (12%) 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., Euclidian distance, cosine similarity and parametric function).

➢ My parametric function:



I concatenate two vectors together, and send into two-layers MLP output one score, which indicate the similarity score the two vector is.

➢ Different distance function of Prototypical network:

Backbone	Meta-Train	Meta-Valid	Episodes	Distance F	Valid acc (%)
Conv4	5-way 1-shot	5-way 1-shot	90000	Euclidian	42.60 ± 0.85
Conv4	5-way 1-shot	5-way 1-shot	90000	Cosine	18.26 ± 0.34

Conv4	5-way 1-shot	5-way 1-shot	90000	Parametric F	46.76 ± 0.87

(Table , same meta setting, diff distance function)

From the experiments, we can found that parametric distance function performance better, but not obvious improve, when meta-training and meta-validating is under the same setting. But I have another experiments below shows that if meta-training have much bigger ways which would let parametric distance function have dominate improve to Euclidian distance.

Backbone	Meta-Train	Meta-Valid	Episodes	Distance F	Valid acc (%)
Conv4	30-way 1-shot	5-way 1-shot	30000	Euclidian	45.97 ± 0.92
Conv4	30-way 1-shot	5-way 1-shot	30000	Cosine	18.26 ± 0.37
Conv4	30-way 1-shot	5-way 1-shot	30000	Parametric F	47.54 ± 0.92

(Table , bigger train-ways, diff distance function)

3. (12%) 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)

Backbone	Meta-Train	Meta-Valid	Episodes	Distance F	Valid acc (%)
Conv4	5-way 1-shot	5-way 1-shot	90000	Parametric F	46.76 ± 0.87
Conv4	5-way 5-shot	5-way 5-shot	90000	Parametric F	48.55 ± 0.90
Conv4	5-way 10-shot	5-way 10-shot	90000	Parametric F	48.50 ± 0.87

From the experiments, we can find that more shot can result better, there are some improvement from 1-shot to 5-shot, with 1.79% valid accuracy improvements, but in my experiment the difference of 5-shot & 10-shot is not improved, I think it is due to the training episodes, 10-shot maybe should need more training episodes to make it converge.

Problem 2: Data Hallucination for Few-shot Learning (50%)

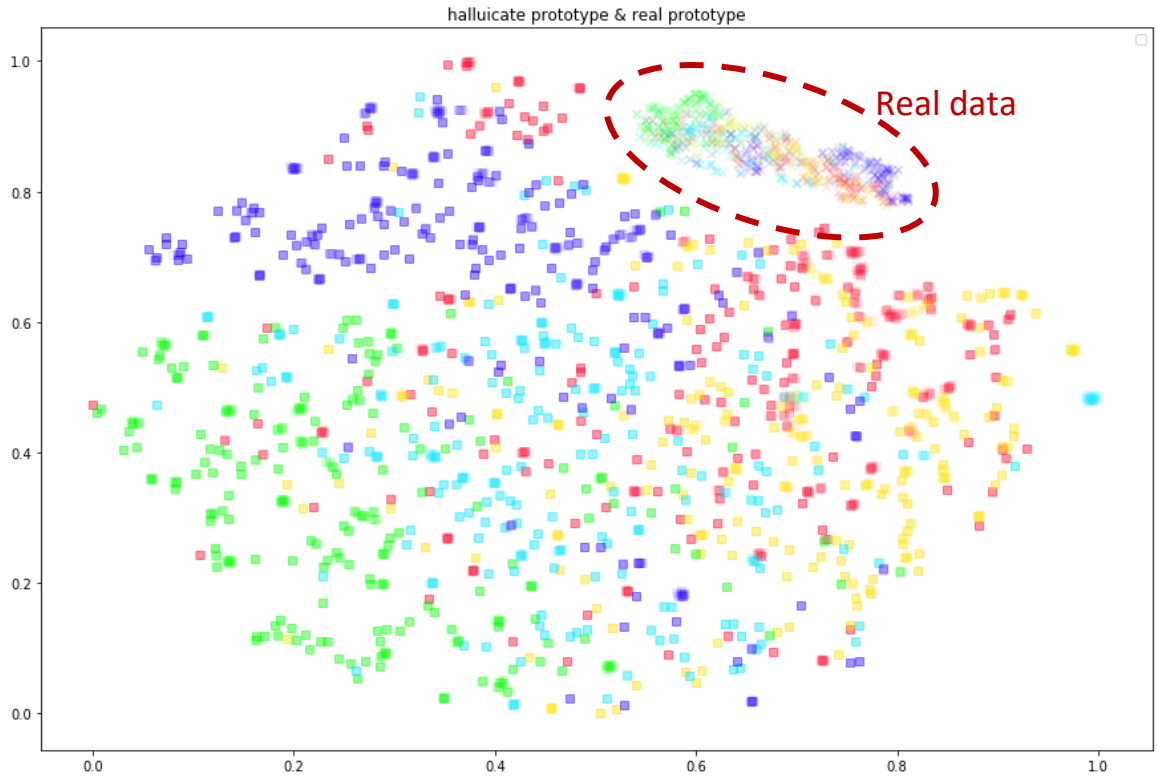
1. (10%) Describe the architecture & implementation details of your model.

Backbone	Meta-Train	Meta-Valid	Haullu_m	Episodes	Distance F	Valid acc (%)
Conv4	30-way 1-shot	5-way 1-shot	20-aug	30000	Parametric F	49.81 ± 0.93

- Validation set accuracy: 49.81 ± 0.93 % 👍

Same as my problem 1 experiment settings: meta-train 30-way 1-shot & meta-valid 5-way 1-shot with 30000 training episodes, and same learning rate, data augmentation, query size, optimizer, distance function. The one with 20-data augment is improved about 4% accuracy compare with naïve support sets. Which can proof that data hallucination do improve performance. I use 10-dimension gaussian noise to produce augment data.

2. (10%) To analyze the quality of your hallucinated data, please visualize the real and hallucinated (training) data in 2D space (with t-SNE).



(Real data are shown as crosses and hallucinated data are shown as rectangle)

[fig2-2.png]

From the t-SNE result, we can find that the hallucinate features is not mix into the real data features. Obviously, there are two clusters between real data & hallucinate data.

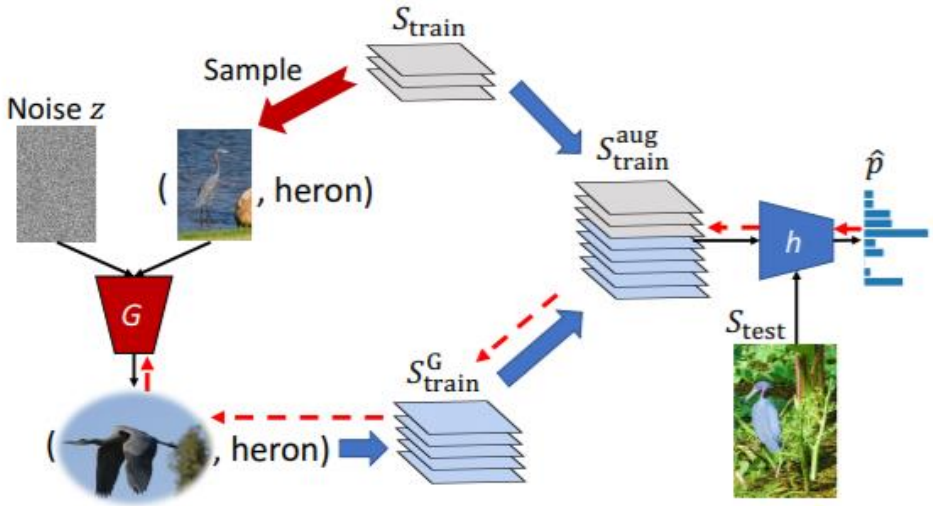
3. (10%) When meta-train and meta-test under the same 5-way 1-shot M-augmentation setting, please report and compare the accuracy with different number of hallucinated data. (M=10, 50, 100)

Backbone	Meta-Train	Meta-Valid	Haullu_m	Episodes	Distance F	Valid acc (%)
Conv4	30-way 1-shot	5-way 1-shot	10-aug	90000	Parametric F	48.69 ± 0.98
Conv4	30-way 1-shot	5-way 1-shot	50-aug	90000	Parametric F	49.20 ± 0.94
Conv4	30-way 1-shot	5-way 1-shot	100-aug	90000	Parametric F	49.52 ± 0.98

Under the same training episodes 90000, I think the number of augmented data can tell the difference to the performance.

4. (5%) Discuss what you have observed and learned from implementing the data hallucination model.

I found that hallucinate do help. The effect of hallucinating is to make augmentations to data. So the model will be more general to deal with different or unseen images. I also observed that under the same episode number, the performance under different number of augmentation data is almost same.



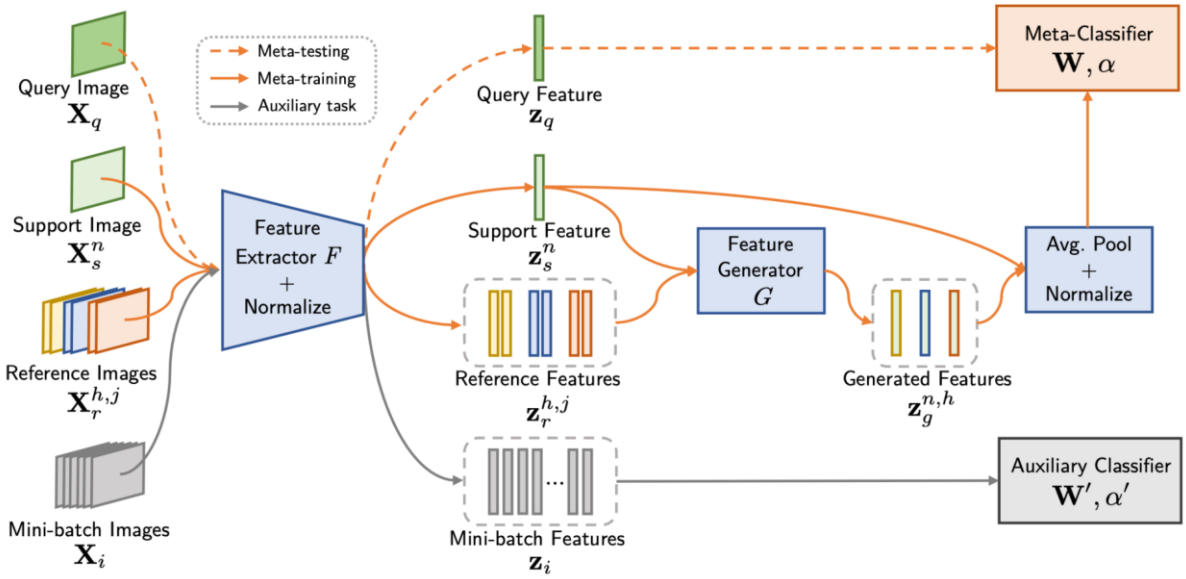
Problem 3: Improved Data Hallucination Model: DTN (20%)

1. (9%) Describe the architecture & implementation details of your model.

Backbone	Meta-Train	Meta-Valid	Haullu_m	Episodes	Distance F	Valid acc (%)
Conv4	30-way 1-shot	5-way 1-shot	20-aug	30000	Parametric F	51.17 ± 0.93

- Validation set accuracy: 51.17 ± 0.93 % 👍

I choose Diversity Transfer Network(DTN) for my improved data hallucination model.

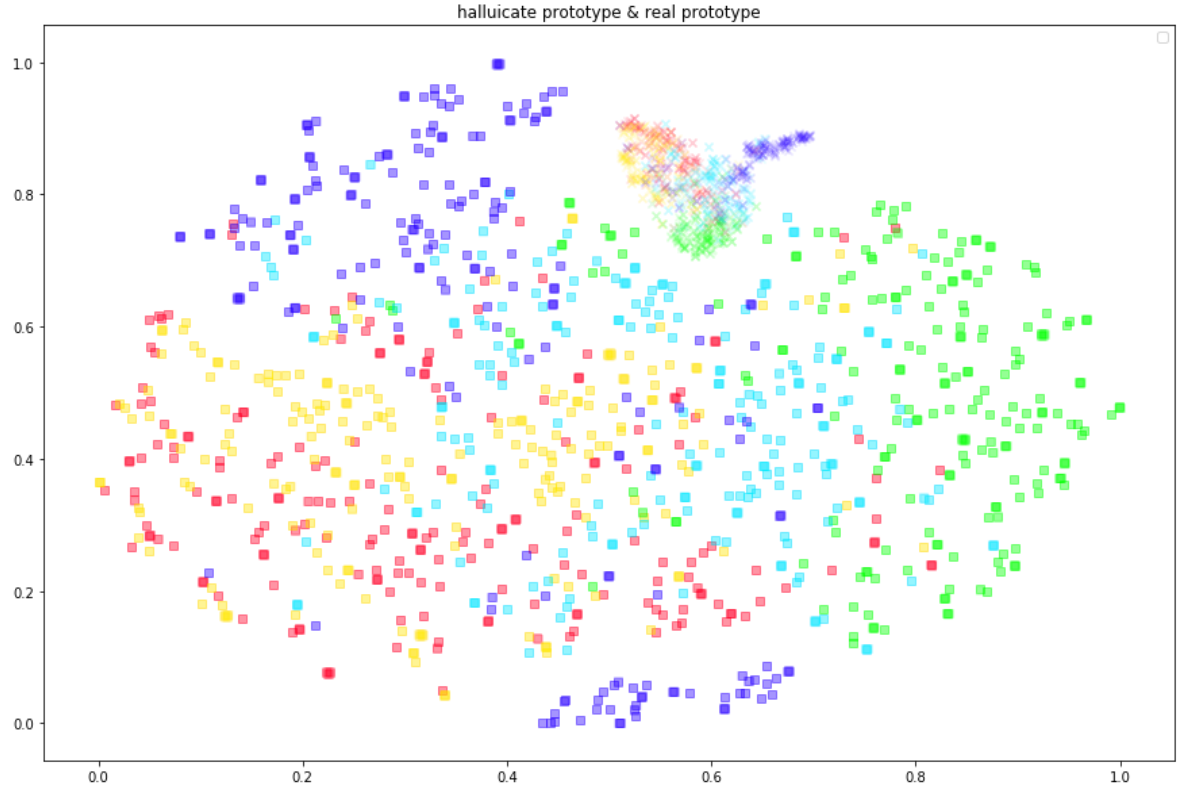


Same as my problem 1 & problem 2 experiment settings: meta-train 30-way 1-shot & meta-valid 5-way 1-shot with 30000 training episodes also 20-data augment, the DTN improved 1.36% performance compare to problem 2 pure data-augment model .

- Implementation details:
  - Training parameters:
    - Training episodes: 100 epochs * 300 batches = 30000 episodes.
    - Learning rate for base: 1e-4
    - Learning rate for meta: 2e-3 (warmup 4500 episodes, cosine annealing)
    - Optimizer: Adam optimizer
  - Data augmentation (same as P1 & P2):

- Random crop(scale=(0.5,1.0))
- Random horizontal flip()
- Random apply p=0.8, Color Jitter(0.8, 0.8, 0.8, 0.2)

2. (2%) To analyze the quality of your hallucinated data, please visualize the real and hallucinated (training) data in 2D space (with t-SNE).



(Real data are shown as crosses and hallucinated data are shown as rectangle)

[fig3-2.png]

3. (4%) 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).

Why the DTN model can be improved. Because for DTN we will have multi-task training , with extra mini-imagenet classification task, it will let model recognize more features of the images. And for my training process, I apply warmup learning rate to my meta-training but image classification without warmup, which will let the model tune on classification task during early stage of the training, and then the

model would become a good feature extractor, which can let us tune on meta-training task.

- Comparison:

	Backbone	Meta-Train	Meta-Valid	Haullu	Episode	Distance F	Valid acc (%)
Naïve-hallu	Conv4	30-way 1-shot	5-way 1-shot	20-aug	30000	Parametric	49.81 ± 0.93
DTN-hallu	Conv4	30-way 1-shot	5-way 1-shot	20-aug	30000	Parametric	51.17 ± 0.93

- Improved 1.36%

[Reference]

- Prototypical Network: https://github.com/yinboc/prototypical-network-pytorch
- Diversity Transfer Network(DTN): https://arxiv.org/pdf/1801.05401.pdf
- DTN pytorch: https://github.com/Yuxin-CV/DTN