Deep Learning for Computer Vision HW4 Report

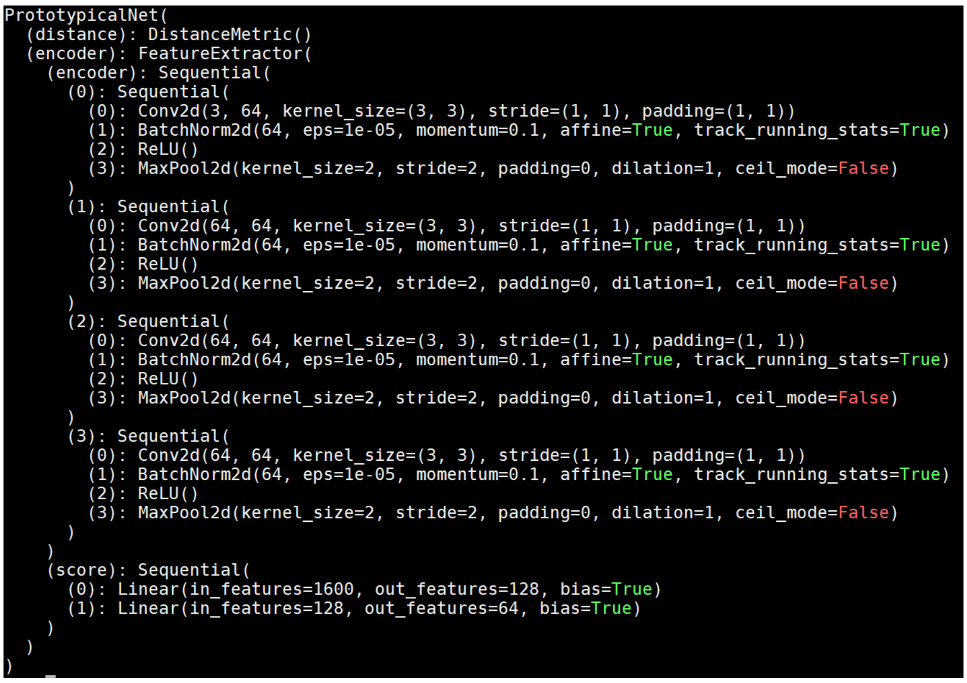
資工所碩二 R08922086 蔡承運

Collaborators: R08922130 丁杰

Problem 1: Prototypical Network

(1) Model architecture & Implementation details

5way – 1shot validation accuracy (from val_testcase.csv): **46.22 +- 0.84%**



```
PrototypicalNet(
  (distance): DistanceMetric()
  (encoder): FeatureExtractor(
    (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)
      )
    )
    (score): Sequential(
      (0): Linear(in_features=1600, out_features=128, bias=True)
      (1): Linear(in_features=128, out_features=64, bias=True)
    )
  )
)
```

Training settings:

✓ Meta-train:
- Episode: 100
- Distance function: Euclidean distance (L2 distance)
- Optimizer: Adam
- Epoch: 50
- Learning rate (lr): 0.001
- Learning rate scheduling: decay lr to lr*0.5 every 20 epoch
- 30 way – 1 shot
- No data augmentation is used

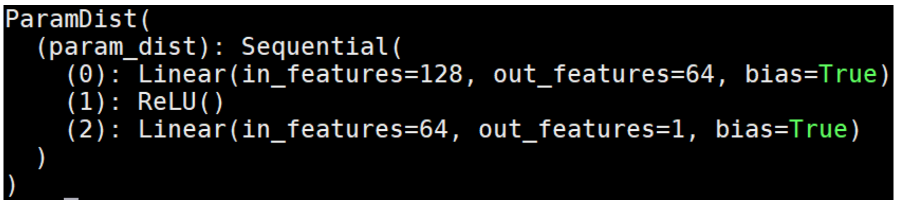
✓ Meta-test:
- Episode: 400
- Distance function: Euclidean distance (L2 distance)
- 5 way – 1 shot
- No data augmentation is used

Other hyper-parameters follow the default values in PyTorch

(2) Effect of different distance functions (meta-train/test are both 5way-1shot)

- Design of parametric function

  I concatenate the prototype of queries and shots to form a new tensor, which is then be passed through the following model to compute the resulting distance.



```
ParamDist(
  (param_dist): Sequential(
    (0): Linear(in_features=128, out_features=64, bias=True)
    (1): ReLU()
    (2): Linear(in_features=64, out_features=1, bias=True)
  )
)
```

- Mean accuracy:

  Training epoch is fixed to 50, seed to 2021, and learning rate to 0.001

Distance metric	Accuracy
Euclidean Distance	41.29 +- 0.78%
Cosine Similarity	37.70 +- 0.75%
Parametric function	38.22 +- 0.81%

- Discussion:

  It is shown that the performance of using Euclidean distance is the best, followed by the parametric function and cosine similarity counterparts. I think this is due to the prototypical network algorithm, as it is meant to project the data onto some latent space. This result complies with the experiment results in the original paper, in which the author conjectured that as cosine distance is not a Bregman divergence, the mixture density estimation in the paper does not hold and thus have poorer estimation.

(3) Effect of different shots (meta-train/test have the same ways and shots)

- Mean accuracy:

  Training epoch is fixed to 50, seed to 2021, and learning rate to 0.001

K	Accuracy
1	41.937%
5	62.330%
10	65.400%

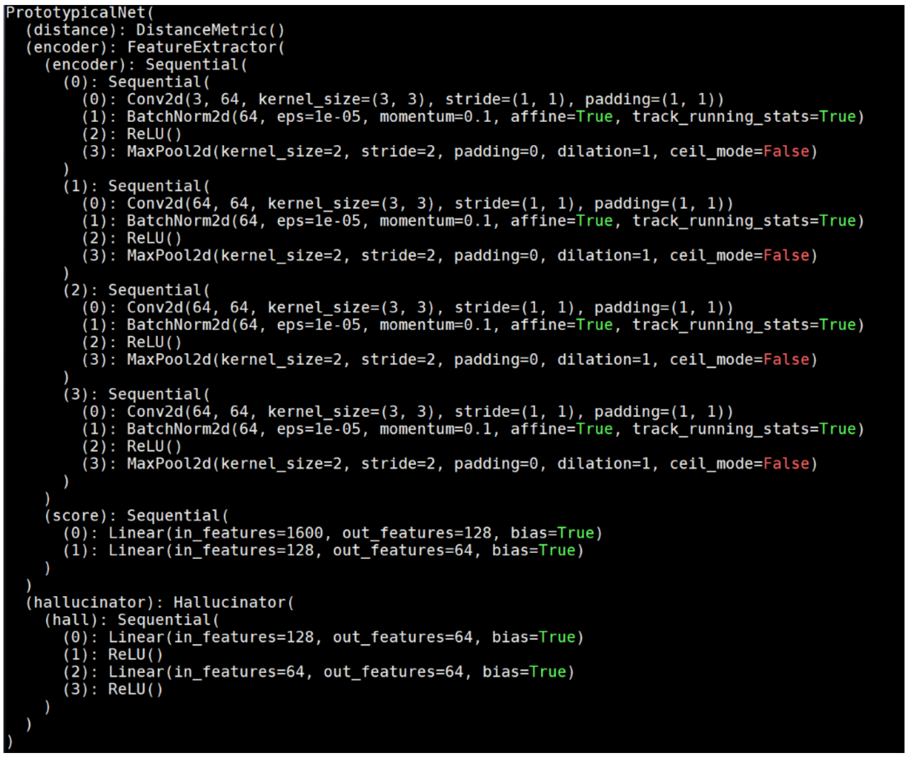
- Discussion:

  When trained under smaller shots in meta-training phase, model is provided with more information from support. The prototypes learned will be more informative and hence the accuracy increases as K (shot) grows.

Problem 2: Data Hallucination for Few-shot Learning

(1) Model architecture & Implementation details

5way – 1shot validation accuracy (from val_testcase.csv): **47.42 +- 0.87%**



```
PrototypicalNet(
  (distance): DistanceMetric()
  (encoder): FeatureExtractor(
    (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)
      )
    )
    (score): Sequential(
      (0): Linear(in_features=1600, out_features=128, bias=True)
      (1): Linear(in_features=128, out_features=64, bias=True)
    )
  )
  (hallucinator): Hallucinator(
    (hall): Sequential(
      (0): Linear(in_features=128, out_features=64, bias=True)
      (1): ReLU()
      (2): Linear(in_features=64, out_features=64, bias=True)
      (3): ReLU()
    )
  )
)
```

Training settings:

50 Epochs with M being 150 in both meta-train and meta-test

✓ Meta-train:

- Episode: 100
- Distance function: Euclidean distance (L2 distance)
- Optimizer: Adam
- Learning rate (lr): 0.001
- Learning rate scheduling: decay lr to lr*0.5 every 20 epoch
- 30 way – 1 shot
- No data augmentation is used

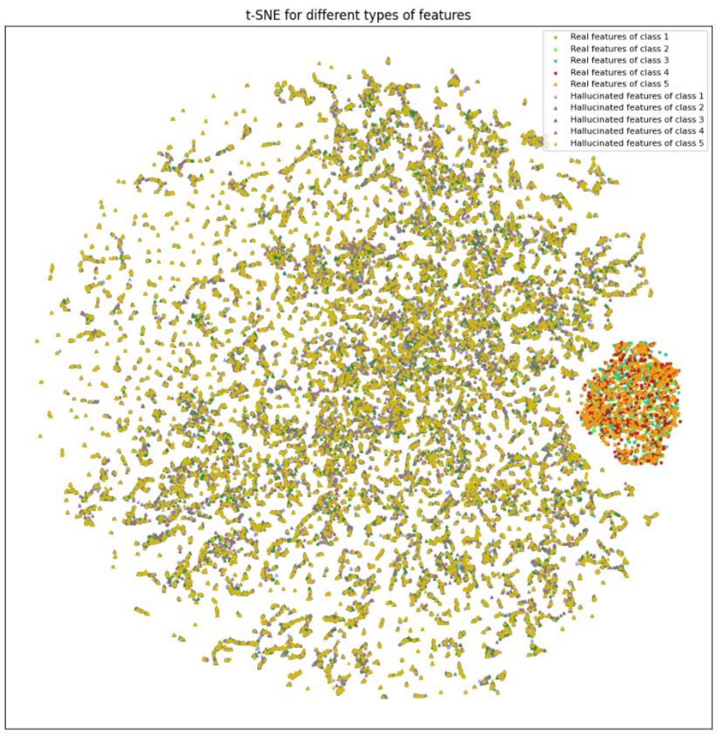
✓ Meta-test:

- Episode: 400
- Distance function: Euclidean distance (L2 distance)
- 5 way – 1 shot
- No data augmentation is used

Other hyper-parameters follow the default values in PyTorch

(2) Visualize the real and hallucinated data in latent space with t-SNE

The hallucinated features are marked using an upper triangle and real features are marked with a cross.



(3) Effect of "M" (meta-train/test under 5way-1shot, M-augmentation setting)

- Mean accuracy:

  Training epoch is fixed to 50, seed to 2021, and learning rate to 0.001

M	Accuracy
10	42.72 +- 0.79%
50	43.35 +- 0.79%
100	43.42 +- 0.78%

- Discussion: The accuracy becomes better as M increases. This is probably owing to the fact that the model has more features to learn from and therefore having better discriminative power in classification.

(4) Discuss what's observed and learned

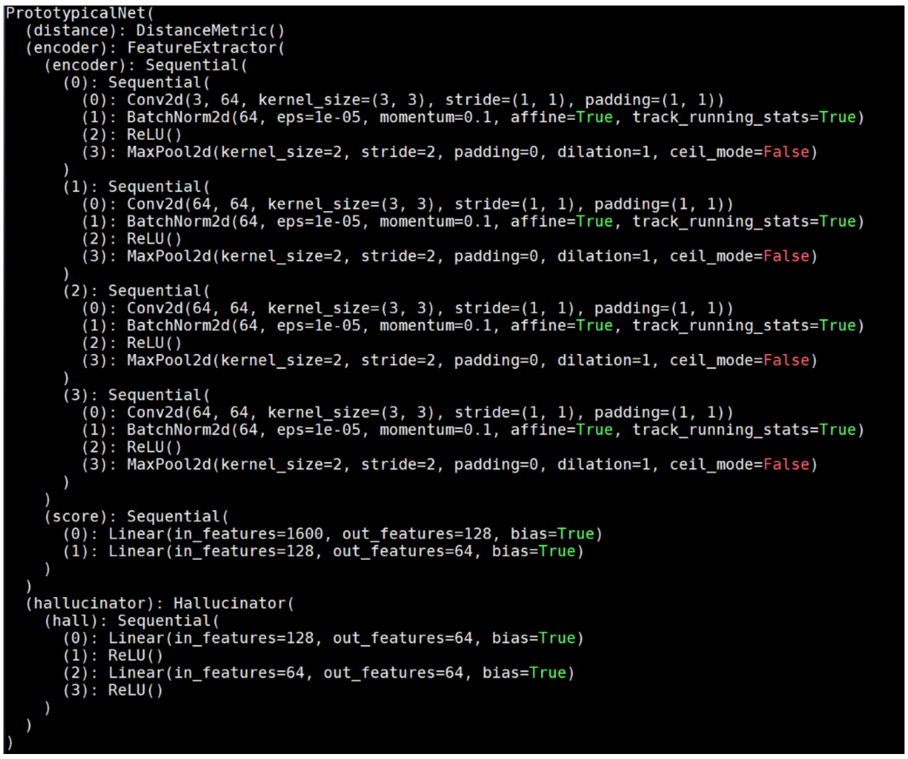
By simply augmenting model with some hallucinated features, the performance boost significantly. This is a useful technique especially when the number of shots is limited. However, the hallucination architecture contains only a generator, which might suffer from lack of diversity and fail to generate informative and diverse features. If we can further add a discriminator, the performance and the quality of the hallucinated features will both be better.

Problem 3: Improved Data Hallucination for Few-shot Learning

(1) Model architecture & Implementation details

5way – 1shot validation accuracy (from val_testcase.csv): **48.08 +- 0.90%**

In this section, I follow the method of data hallucination, the popular solution regarding few-shot learning, as it directly solves the problem of data scarcity. One real feature is selected per class as seed. These seeds are concatenated with a random noise sampled from Gaussian distribution, meaning a small fluctuation is added to the learned features. Following the similar process, we can hallucinate from the hallucinated features again to get much more diverse features. All the generated features are combined together with the real features, which is averaged to be the prototype for each class. The remaining process follows the algorithm in the Prototypical Network.



```
PrototypicalNet(
  (distance): DistanceMetric()
  (encoder): FeatureExtractor(
    (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)
      )
    )
    (score): Sequential(
      (0): Linear(in_features=1600, out_features=128, bias=True)
      (1): Linear(in_features=128, out_features=64, bias=True)
    )
  )
  (hallucinator): Hallucinator(
    (hall): Sequential(
      (0): Linear(in_features=128, out_features=64, bias=True)
      (1): ReLU()
      (2): Linear(in_features=64, out_features=64, bias=True)
      (3): ReLU()
    )
  )
)
```

Training settings:

50 Epochs with M being 150 in both meta-train and meta-test

✓ Meta-train:

- Episode: 100
- Distance function: Euclidean distance (L2 distance)
- Optimizer: Adam
- Learning rate (lr): 0.001
- Learning rate scheduling: decay lr to lr*0.5 every 20 epoch
- 30 way – 1 shot
- No data augmentation is used

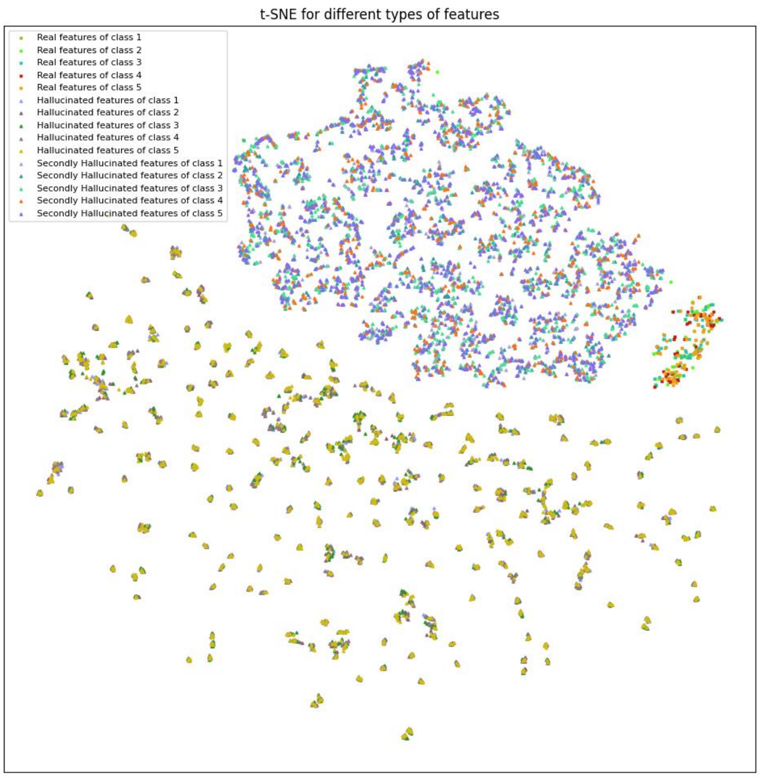
✓ Meta-test:

- Episode: 400
- Distance function: Euclidean distance (L2 distance)
- 5 way – 1 shot
- No data augmentation is used

Other hyper-parameters follow the default values in PyTorch

(2) Visualize the real and hallucinated data in latent space with t-SNE

As shown in the t-SNE figure below, the features of different types are grouped together and thus add diversity to the generated features.



(3) Try to explain why the improved model performs better

Simply hallucinating features rom the real features with some random noise do provide additional diversity to the model. If we directly generate features again from the same set of real features, which might have similar (close) features with each other and have less diversity, the resulting hallucinated features might be similar to the one in the first set of hallucinated features, which losses the essence of hallucinating features as we want more diverse features for the model to learn better from limited data. We can also see the results from the figure in Problem 3.2 and in Problem 2.2 are features that are more diverse in the latent space.