DLCV HW4 Report

B07901039 電機四 劉知穎

Problem 1: Prototypical Network

1. Describe the architecture & implementation details of your model.

(1) Validation accuracy (on testcase): 47.11 +- 0.85 %

(2) Model architecture

The model architecture is shown in Figure 1, it is same as required in hw4 introduction ppt p.14. The input of the model is images of size (3, 84, 84), and the output of the model for each image is 1600-dim feature vector.

```
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)
    )
  )
)
```

Figure 1: Model architecture of prototypical network

(3) Implementation details

- N_shot_train=1
- N_way_train=20
- N_query_train=15
- N_episode_per epoch_train=100
- N_shot_val=1
- N_way_val=5
- N_query_val=75
- N_episode_per_epoch_val=100
- N_epoch=150
- Distance function: Euclidean distance
- Optimizer: Adam, Learning rate: 1e-3
- 20-way 1-shot in meta-train phase, and 5-way 1-shot in meta-test phase

Model can be trained better by using larger N_shot at meta-train phase. With

other setting remaining same and testing on 5-shot validation testcase, using 5-shot at meta-train phase achieves accuracy 43.88 +- 0.80 %, 20-shot achieves accuracy 47.11 +- 0.85 %, and 25-shot achieves accuracy 46.71 +- 0.82 %. Since 20-shot and 25-shot have similar performance on meta-test phase, I chose the model trained with 20-shot to be the final model.

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).

   (1) The accuracy is computed on the validation testcase.

Euclidean distance	Cosine similarity	Parametric function
43.88 +- 0.80 %	39.01 +- 0.79 %	43.36 +- 0.82 %

   (2) Parametric function design

   The parametric function design is shown in Figure 2. I concatenate the feature vectors with each prototype sequentially, feed the 3200-dim concatenated vector into the parametric model, and the output will be a scalar representing how the feature vector is 'similar' to the prototype. The parametric model architecture is 3 linear layers, with output dimension 3200, 1000, 1 respectively, and the activation functions are ReLU.

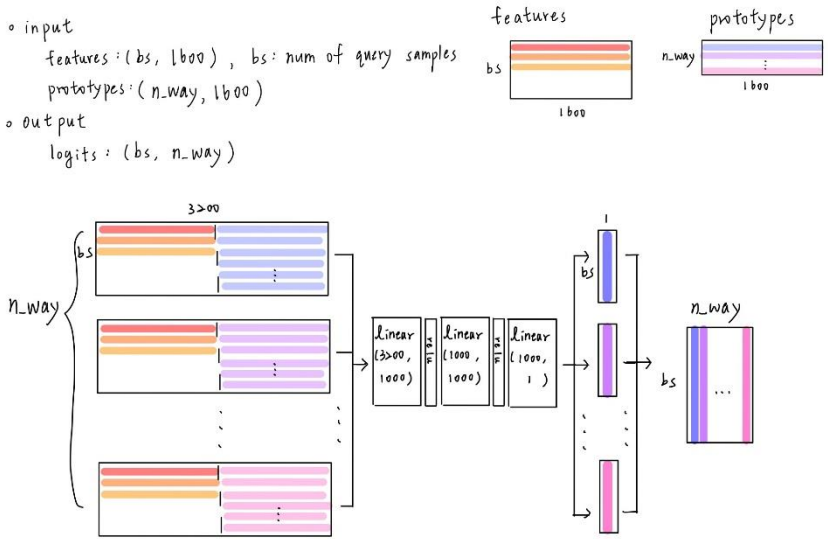

Figure 2: Parametric function design

Note that different prototypes share the same parameter in Figure 2. I think that rather than each prototype having its own parameter, sharing parameter is more reasonable for distance function. If I design the parametric function as shown in Figure 3. Simply concatenate a 1600-dim feature vector with N-way 1600-dim prototypes, input the 1600*(N-way + 1)-dim vector into the parametric model, and output N-way-dim logit. In this case, we learn separate weights for each prototype. In the end, the model fails to learn anything (the validation accuracy is about 0.2).

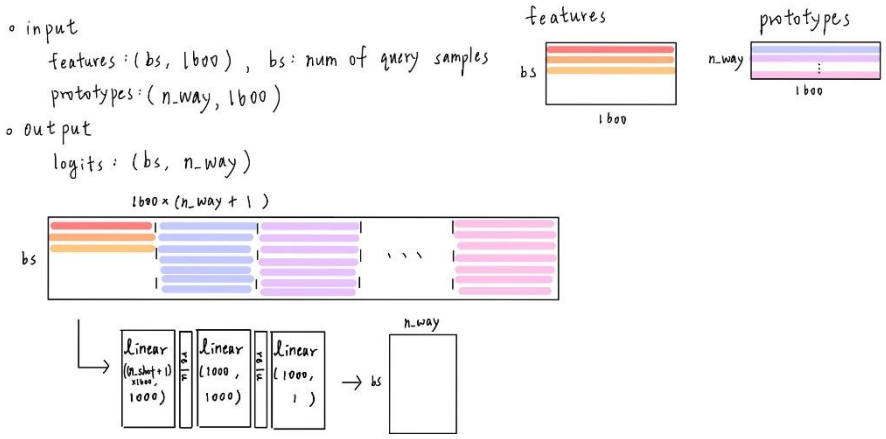

Figure 3: Parametric function design (fail)

(3) Discussion

a. Euclidean distance obviously outperforms cosine similarity. This result is identical to the result in Snell, Swersky, and Zemel [1]. [1] states that "computing class prototypes is more naturally suited to Euclidean distances since cosine distance is not a Bregman divergence".

b. Parametric function and Euclidean distance have similar performance. I believe that parametric function may outperform Euclidean distance by better designing the model architecture, preprocess input of the model, or train for longer epochs.

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)

(1) Randomly sample 600 episodes. In each episode, the accuracy is computed over 5 * 15 query data, where each way has 15 query data. Each case is train with 5-way K-shot meta-train setting on training dataset, and test with 5-way K-shot meta-test setting on validation dataset. Other settings remain same.

5-way 1-shot	5-way 5-shot	5-way 10-shot
43.87 +- 0.79 %	63.78 +- 0.69 %	68.78 +- 0.63 %

(2) Discussion

Models trained with larger K-shot at both meta-train and meta-test phases perform better. The result is reasonable since model can computed a more robust prototype for each class with more support data.

Problem 2: Self-Supervised Pre-training for Image Classification

1. Describe the implementation details of your SSL method for pre-training the ResNet50 backbone.
   - SSL method: BYOL (use packages provided in [2])
   - Data augmentation:RandomGrayscale, RandomHorizontalFlip, RandomResizedCrop, Normalize, and randomly apply ColorJitter, GaussianBlur
   - Learning rate schedule: first train 60 epochs with learning rate 1e-3, then train 30 epochs with learning rate 1e-4. Use the model from the last epoch.
   - Optimizer: Adam, betas = (0.9, 0.999)
   - Batch size: 32
2. Conduct the Image classification on Office-Home dataset as the downstream task for your SSL method. Also, complete the following Table and compare the results.

Setting	Pre-training (Mini-ImageNet)	Fine-tuning (Office-Home dataset)	Validation accuracy
A	X	Train full model	0.4828
B	With label	Train full model	0.5197
C	Without label	Trian full model	**0.4187**
D	With label	Fix backbone	0.3177
E	Without label	Fix backbone	0.1010

   All experiments are conducted in the following setting:
   - Epoch: 100, save the model with the best validation accuracy
   - Learning rate scheduler: 1e-3 for epoch 1-50, and 1e-4 for epoch 51-100
   - Optimizer: Adam, betas = (0.9, 0.999)
   - Batch size: 64
   - Data augmentation: RandomHorizontalFlip, RandomRotation, ColorJitter, RandomPerspective
   - Classifier: linear(2048, 65)
3. Discussion

   (1) Pretraining model with labels perform better than pretraining model

without labels. It is reasonable because the former case may have a better model initialization.

(2) At fine-tuning phase, training whole model performs better than only training the classifier. It is reasonable because the former case can adjust the backbone based on the downstream dataset.

(3) Setting A (train from scratch) performs better than setting C (train from a self-supervising pretrained model). It is counter intuitive. The training curves of the two settings are shown in Figure 4.

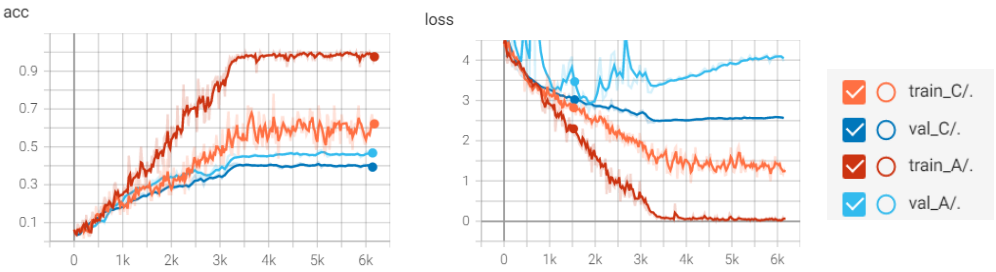

Figure 4: Training curve of setting A and C

The validation accuracy of setting C (with pretrained model) is slightly greater the validation accuracy of setting A (train from scratch) at the early 8000 iterations. However, after 8000 iterations, the performance of setting A surpass that of setting C both on training and validation dataset.

[3] states that under strong data augmentation or large dataset, pre-training may be harmful to model performance. Therefore, I conducted setting A and C experiments again under no data augmentation, and simply train 50 epochs with learning rate 1e-3. The result is presented in the following table.

Setting	Validation accuracy
A	0.3276
C	0.2414

Setting A still performs better than setting C in no data augmentation case.

Explanation:

In pretraining phase, the training loss oscillated between 1.0 and 0.1 even after 90 epochs. The validation loss is more unstable, it oscillated between 3.0 and 0.1. (Note that I only use validation loss to check performance, I didn't use it to choose the model). I think that the pretrained model fail to learn image information, so pretraining in this case worsen the model's performance in fine-tuning phase.

Reference

1. Jake Snell, Kevin Swersky, and Richard Zemel. 2017. Prototypical networks for few-shot learning. In Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17). Curran Associates Inc., Red Hook, NY, USA, 4080–4090.
2. https://github.com/lucidrains/byol-pytorch
3. Kaiming He, Ross Girshick, Piotr Doll´ar. 2018. Rethinking ImageNet Pre-training. ICCV.