DLCV HW4 Report

B07502071 機械三 陳志臻

P1、Prototypical Network

1. implementation details

Accuracy: 43.99 +- 0.83 %

Prototype Network architecture:

conv: { nn.Conv2d(in\_channels, out\_channels, 3, padding=1),

bn,

nn.ReLU(),

nn.MaxPool2d(2) } \* 4

mlp: { nn.Linear(1600, 800),

nn.ReLU(),

nn.Dropout(),

nn.Linear(800, 800),

nn.ReLU(),

nn.Dropout(),

nn.Linear(800, 400) }

Optimizer: optimizer = torch.optim.AdamW(model.parameters(), lr=1e-2,

betas=(0.9, 0.999), eps=1e-08, weight\_decay=1e-4)

meta-train setting: 5 ways 5 shots

meta-test setting : 5 ways 1 shot

distance function: parametric function

training epoch: 200

lr-scheduler: none

transforms: none

1. distance function (all in 5 way 1 shot meta learning)

Euclidean distance: Accuracy: 38.94 +- 0.78 %

cosine similarity: Accuracy: 42.62 +- 0.86

parametric function: Accuracy: 43.36 +- 0.86 %

parametric function model architecture:

nn.Sequential(

nn.Linear(800, 400),

nn.ReLU(),

nn.Dropout(),

nn.Linear(400, 1)

)

discuss:

All these three tasks were trained on the setting of 5 ways 1 shot meta-train and 5 ways 1 shot meta-test. All of them were trained for 200 epochs. We can find that if we applied the Euclidean distance function, it would get a worse performance than the other two distance functions. The parametric function got the best performance. I think it is make sense because we use a prototype network to project our images to a feature space, we should not calculate all the feature distances between any two images in the same way.

1. N-way K shot

K=1: Accuracy: 43.36 +- 0.86 %

K=5: Accuracy: 43.99 +- 0.83 %

K=10: Accuracy: 43.05 +- 0.86 %

discuss:

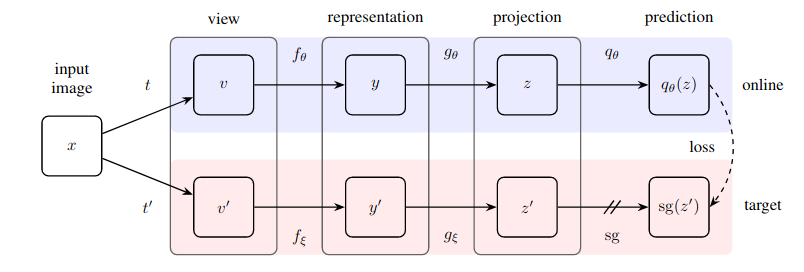
At first I thought that the 5-way 10-shot meta-learn task would get the best performance. I think that’s because I need to train for more steps, it may be harder for training a task with higher shots. We can also find that 5-way 1-shot meta-learn task converge sooner. I guess if I trained it for more steps, a 5-way 10-shot meta-learn task can get a better result.

P2、Self-Supervised Pre-training for Image Classification

1. implementation details

backbone model: resnet50

learner: BYOL package



Classifier model: nn.Linear(1000, 65)

data augmentation for pretraining:

filenameToPILImage,

transforms.ToTensor(),

transforms.Resize((128, 128)),

transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]

data augmentation for fine turning:

filenameToPILImage,

transforms.AutoAugment(),

transforms.ToTensor(),

transforms.Resize((128, 128)),

transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]

Optimizer: optimizer = torch.optim.AdamW(model.parameters(), lr=2e-5, betas=(0.9, 0.999), eps=1e-08, weight\_decay=1e-4, amsgrad=False)

Loss function: nn.CrossEntropyLoss()

Lr-scheduler: get\_cosine\_schedule\_with\_warmup

Fine turning Training epoch: 100

1. Result table

|  |  |  |  |
| --- | --- | --- | --- |
| Setting | Pre-training (Mini-ImageNet) | Fine-tuning (Office-Home dataset) | Classification accuracy on valid set (Office-Home dataset) |
| A |  | Train full model (backbone + classifier) | 0.12 |
| B | w/ label (TAs have provided this backbone) | Train full model (backbone + classifier) | 0.24 |
| C | w/o label (Your SSL pre-trained backbone) | Train full model (backbone + classifier) | 0.37 |
| D | w/ label (TAs have provided this backbone) | Fix the backbone. Train classifier only | 0.17 |
| E | w/o label (Your SSL pre-trained backbone) | Fix the backbone. Train classifier only | 0.18 |

Table[1]

1. Discuss and analyze

I trained my models on P100 GPU, it cost almost 7 hours to train one task. Most of the training tasks would converge at about 60th epoch. Before I got the result of the table[1], I guessed that the B set should get the highest accuracy. However, that was wrong. The C set got the highest accuracy. I think the reason which caused this situation is: model pretrained on Mini dataset with labels will overfit on the dataset. If we pretrained in self-supervised learning, it wouldn’t overfit on training dataset. Besides, we can also find that if we fixed the backbone model, the result could be worse than training full model.

Reference:

<https://github.com/lucidrains/byol-pytorch>

<https://github.com/sicara/easy-few-shot-learning>

<https://github.com/kai860115/DLCV2020-FALL/tree/main/hw4>