DLCV HW4 Report

Shuo-En Chang
R08922A02

December 21, 2020

1 Prototypical Network

1.1 My implement of Prototypical Network.

In the Convnet4, first I use 4 Conv2d layers with out_channel 64 as my encoder, where each filter size is set to 3×3. There are BatchNorm between each of them and ReLU is used as the activation function. I use max-pooling as my downsampling method. After the encoder are 2 linear layers for converting features into metric space, which use 100 as the space dimension.

The accuracy on validation set under 5-way 1-shot setting is $46.79 \pm 0.85\%$.

Following are some hyperparameters using in the training process.

- Metric space dimension: 100
- Training episodes for each epoch: 1,000
- Training epoch: 50
- distance function: Euclidean
- Optimizer: Adam
- Learning rate: 0.0005
- Learning rate schedule: Decays 0.75 every 5 epochs
- Data preprocess: Normalize each picture
- Meta setting: 5-way 1-shot, with N_query 15

```
Convnet4(
  (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): Sequential(
  (0): Flatten()
  (1): Linear(in_features=1600, out_features=500, bias=True)
  (2): ReLU(inplace=True)
  (3): Linear(in_features=500, out_features=100, bias=True)
 )
)
```

1.2 The accuracy with using 3 different distance function.

The parametric function is just three linear layers with ReLU between them. The input of forward pass for this model is (x, y), where the size of x is (n_class, f_dim), and the size of y is (n_query, f_dim). Both x and y will first be expanded to (n_query, n_class, f_dim) and then concatenated along 'f_dim', which will become (n_query, n_class, f_dim*2). After this function, the output size is (n_query, n_class, 1), which can be seen as the distance between each query and prototypes of each class.

And I find out that parametric function can get better performance compared to cosine similarity but not as good as Euclidean distance. Which can get the same conclusion as Snell *et al.*[3] mentioned that Euclidean distance is the best choice for distance function.

```
Parametric(
  (net): Sequential(
    (0): Linear(in_features=200, out_features=100, bias=True)
    (1): ReLU()
    (2): Linear(in_features=100, out_features=100, bias=True)
    (3): ReLU()
    (4): Linear(in_features=100, out_features=1, bias=True)
  )
)
```

	Euclidean	Cosine	Parametric
Accuracy	46.79%	45.32%	46.08%

1.3 The accuracy with different shots.

In this part, I use random generated episode as meta-test input due to different shots. In the experience, more shots will get higher accuracy. I think this is easy to understand because more shots mean we can see more data in meta-train. Therefore, the prototype of each class is more representative.

	1-shot	5-shot	10-shot
Accuracy	47.4%	61.2%	65.7%

2 Data Hallucination for Few-shot Learning

2.1 My implement of Data Hallucination.

Convnet4 has the same architecture as in problem 1. The Hallucination model is simply 3 linear layers with ReLU between each layer and has the same out_feature.

The accuracy on validation set under 5-way 1-shot 10-augmentation setting is $49.00 \pm 0.89\%$.

Following are some hyperparameters using in the training process.

- Metric space dimension: 100
- Training episodes for each epoch: 1,000
- Training epoch: 50
- distance function: Euclidean
- Optimizer: Adam
- Learning rate: 0.0005
- Learning rate schedule for Convnet4: Decays 0.5 every 5 epochs
- Learning rate schedule for Hallucination: Decays 0.5 every 5 epochs
- Data preprocess: Normalize each picture
- Meta setting: 5-way 1-shot 10-augmentation, with N_query 15

```
Convnet4(
  (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): Sequential(
    (0): Flatten()
    (1): Linear(in_features=1600, out_features=500, bias=True)
    (2): ReLU(inplace=True)
```

```
    (3): Linear(in_features=500, out_features=100, bias=True)
  )
)

Hallucination(
  (net): Sequential(
    (0): Linear(in_features=200, out_features=100, bias=True)
    (1): ReLU(inplace=True)
    (2): Linear(in_features=100, out_features=100, bias=True)
    (3): ReLU(inplace=True)
    (4): Linear(in_features=100, out_features=100, bias=True)
  )
)
```

2.2 Visualize the real and hallucinated data.

Within hallucinated data, most of them are very close to each other but still distinguishable. On the other hand, real data and hallucinated data have different distribution although they are in the same class. Due to the above, we can say that the Hallucination model actually successfully learned how to generate hallucination data. And the hallucination data is indeed different from the real data.

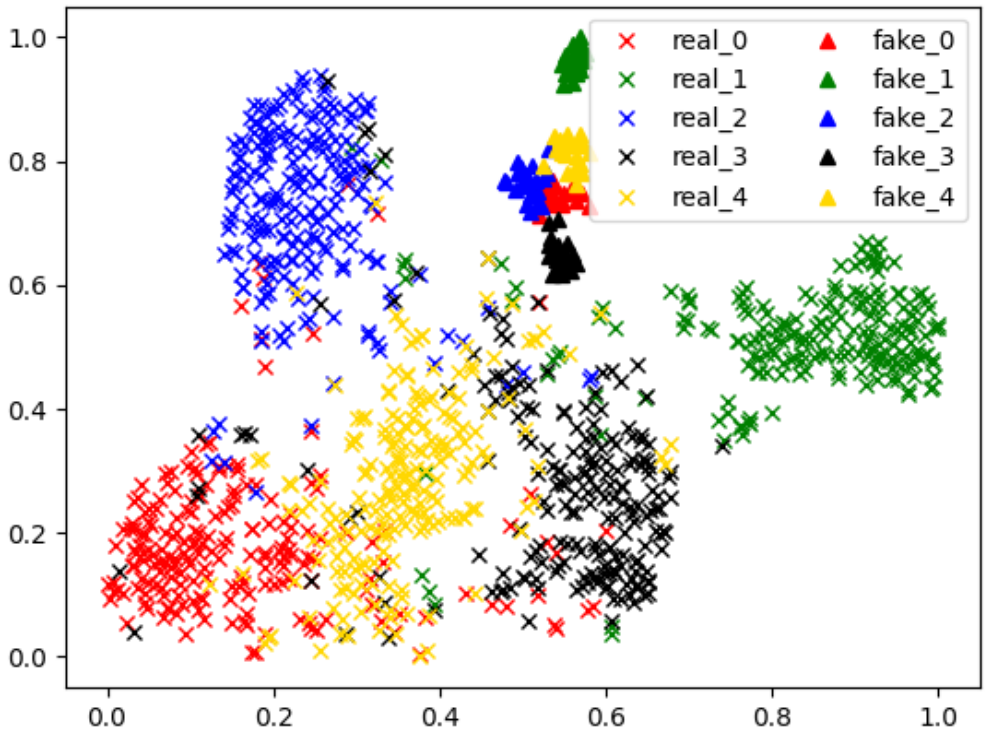

Figure 1: Visualize with real and hallucinated data

2.3 The accuracy with different number of hallucinated data.

I use the same training process with only different M-augmentation in this part. In the experience, the larger M leads to higher accuracy. I think it does make sense because that means the model can see more hallucination data, which somehow is representative of each class. But the premise is that most of the hallucination data they produce can be distinguishable. Otherwise, it will only become noise and lead to worse performance.

Table 1: Compared between different number of hallucinated data

	m=10	m=50	m=100
accuracy	49.00%	49.23%	49.70%

2.4 Observed and learned from implementing the data hallucination model.

I think the augmentation method here is really creative and effective. Intuitively, it is not easy to say that we can learn how to classifier from noise data, using another network to generate from noise can help us doing so. I think this is the situation where GAN's[2] main idea can be used in other task. And surprisingly, even it use only one real data as the seed for generate fake data, it can help the classification task to have a better performance.

3 Improved Data Hallucination Model

3.1 My implement of Improved Data Hallucination Model.

The only change for improvenment is add a Discriminator in meta-train. The main idea is come from WGAN[1]. Expected the origin training pipeline, here will add another training process same as WGAN[1] after it.

The Hallucination model can be viewed as generator, and discriminator is simply 3 linear layers with LeakyReLU between each layers. So, both of them can just use the same training process as in WGAN[1], and use the setting of weight clipping with a limit [-0.001, 0.001]. Note here, it doesn't need the discriminator during meta-testing. It just uses as the guide for Hallucination model during training process.

The accuracy on validation set under 5-way 1-shot 10-augmentation setting is $49.82 \pm 0.91\%$.

Following are some hyperparameters using in the training process. (Same as problem 2)

- Metric space dimension: 100
- Training episodes for each epoch: 1,000
- Training epoch: 50
- distance function: Euclidean
- Optimizer: Adam
- Learning rate: 0.0005
- Learning rate schedule for Convnet4: Decays 0.5 every 5 epochs
- Learning rate schedule for Hallucination: Decays 0.5 every 5 epochs
- Learning rate schedule for Discriminator: Decays 0.5 every 5 epochs
- Data preprocess: Normalize each picture
- Meta setting: 5-way 1-shot 10-augmentation, with N_query 15

```
Convnet4(
  (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): Sequential(
    (0): Flatten()
    (1): Linear(in_features=1600, out_features=500, bias=True)
    (2): ReLU(inplace=True)
    (3): Linear(in_features=500, out_features=100, bias=True)
  )
)

Hallucination(
  (net): Sequential(
    (0): Linear(in_features=200, out_features=100, bias=True)
    (1): ReLU(inplace=True)
    (2): Linear(in_features=100, out_features=100, bias=True)
    (3): ReLU(inplace=True)
    (4): Linear(in_features=100, out_features=100, bias=True)
  )
)

# add Discriminator in training phase
Discriminator(
  (net): Sequential(
    (0): Linear(in_features=100, out_features=50, bias=True)
    (1): LeakyReLU(negative_slope=0.01)
    (2): Linear(in_features=50, out_features=50, bias=True)
    (3): LeakyReLU(negative_slope=0.01)
    (4): Linear(in_features=50, out_features=1, bias=True)
  )
)
```

3.2 Visualize the real and hallucinated data.

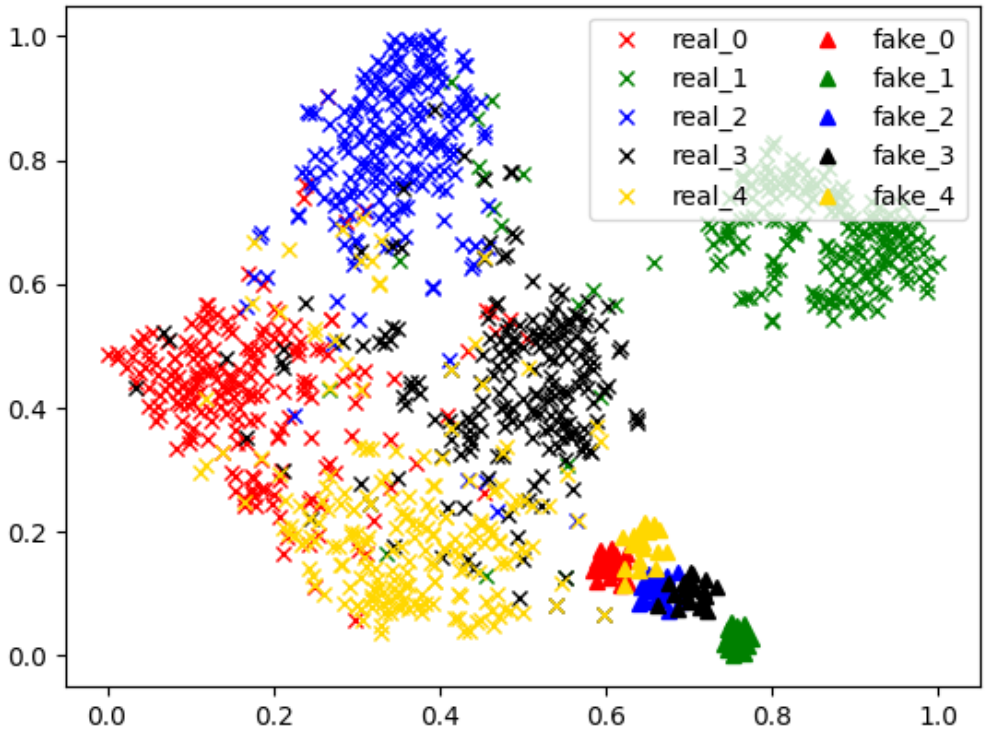

Figure 2: Visualize with real and hallucinated data

3.3 Explain why the improved model performs better than Problem 2.

Compared to Figure 1, I think the distribution in hallucinated data is more likely to be the distribution in real data. For example, in real data, green is closest to black. But in Figure 1, black is farthest to green in hallucinated data. In the same situation, black is nearest to green in Figure 2. Other examples can be found between red and green. Therefore, it can be concluded that the improved model has better performance than the original, possibly because we have better hallucinated data.

4 Collaboration

I had discussed this assignment with Tzu-Hsuan Weng(R08922144), Jie Ting(R08922130), Austin Tsai(R08922086).

References

[1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

[2] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[3] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *Advances in neural information processing systems*, pages 4077–4087, 2017.