NTU AMMAI 2021 Spring HW#1

Face Verification

B07902055 謝宗晅

- Baseline model:
  - SphereFace20a
    - code reference
    - paper reference
- Modifications:
  - Remove A-Softmax Loss, and change it into Cross Entropy Loss (original softmax loss)
  - CosFace
    - code reference
    - paper reference

Execution Commands

```
# training
# model_name: [sphereface, sphereface_base, sphereface_cos, sphereface_arc]
#   sphereface: original version
#   sphereface_base: CrossEntropyLoss
#   sphereface_cos: CosFace Loss
python3 main.py --mode train [--model model_name]
# testing on close setting
python3 main.py --mode close [--model model_name]
# testing on open setting
python3 main.py --mode open [--model model_name]
```

Folder Structure

Please make sure the folder structure is like below so that training and testing can be executed correctly.
(For demo execution, please download models through this link)

```
├── data
│   ├── test
│   │   ├── closed_landmark.txt
│   │   ├── closed_set
│   │   ├── open_landmark.txt
│   │   └── open_set
│   └── train
│       ├── A
│       ├── APDlandmark.txt
│       ├── B
│       └── C
├── genLandmark.py
├── main.py
├── sphereface_CELoss
│   └── models
│       └── *.pth
├── sphereface_CosLoss
│   └── models
│       └── *.pth
└── sphereface_pytorch
    └── models
        └── *.pth
```

Description of Loss Functions

(Reference: 人脸识别合集 | 10 ArcFace解析)

A-Softmax Loss

$(\|W_i\| = 1, b_i = 0)$

$$L = \frac{1}{N}\sum_i -\log\left(\frac{e^{\|\mathbf{x}_i\|\cos(m\theta_{y_i,i})}}{e^{\|\mathbf{x}_i\|\cos(m\theta_{y_i,i})} + \sum_{j\neq y_i} e^{\|\mathbf{x}_i\|\cos(\theta_{j,i})}}\right)$$

CosFace Loss

$(\|W\| = 1, \|x\| = 1)$

$$L = \frac{1}{N}\sum_i -\log\left(\frac{e^{s(\cos(\theta_{y_i,i})-m)}}{e^{s(\cos(\theta_{y_i,i})-m)} + \sum_{j\neq y_i} e^{s\cos(\theta_{j,i})}}\right)$$

ArcFace Loss

$(\|W\| = 1, \|x\| = 1)$

$$L = \frac{1}{N}\sum_i -\log\left(\frac{e^{s\cos(\theta_{y_i}+m)}}{e^{s\cos(\theta_{y_i}+m)} + \sum_{j\neq y_i} e^{s\cos(\theta_j)}}\right)$$

Training

Before training, I used MTCNN to get facial landmarks of each image and write them into `./data/train/APDlandmark.txt` (for reproduction on other dataset, please refer to `genLandmark.py`).
And if there are more than one face in an image, the face with highest confidence score would be chose.
During training, information in `./data/train/APDlandmark.txt` are used to align the faces. Aligned faces are of size (112, 96). Aligned faces are then fed into our model.
All of the models are optimized by `SGD` with momentum 0.9 and weight_decay $5e-4$.
For all the models, initial learning rate is 0.01, and the model can converge in 30 epochs (actually 20 is sufficient).

Inference

Before inference, MTCNN is also used to retrieve facial landmarks for each test image. Also, if there are more than one face, the face with highest confidence score would be chose.
On close set testing, I directly compare the id of testing faces without calculating the similarity of features.
On open set testing, the feature of faces are used to calculate the similarity. Whether two faces are the same is determined by a threshold: if similarity > threshold, then they are the same, and vice versa.

Performance of Each Model

(Training time is only for reference, the situation during their training might not be the same.)

Model	Training Accuracy	Closed-set Accuracy	Open-set Accuracy (Threshold)	Training Time
sphereface	81.89%	74.8%	79.6% (0.4)	80 mins
sphereface (CrossEntropyLoss)	97.57%	70.7%	79.6% (0.3)	28 mins
sphereface (CosFace Loss)	99.04%	77.6%	81.4% (0.25)	30 mins (for highest testing performance, epoch 21) 2.5 hr (for highest training accuracy, 120 epoch)

Experimantal Results

Exp1: Close Setting Inference Mode

In close setting, we directly get the ID of output and therefore don't need to compare the similarity of output

feature vectors. In this section, we evaluate the performance of these methods:

- directly comparing ID
- calculate the similarity of output feature vectors

Model	Performance (closed-set)
sphereface	74.8%
sphereface (feature)	77.1%
sphereface_CELoss	70.7%
sphereface_CELoss (feature)	69.8%
sphereface_cos	77.6%
sphereface_cos (feature)	78.4%

Exp2: With or Without Annealing Optimization
Orignially, my modification on CosFace Loss is without annealing optimization. After adopting annealing optimization, the model can learn faster:

With or Without Annealing Optimization	Epoch 1 Accuracy	Epoch 10 Accuracy	Time Needed to Achieve More Than 75% Accuracy
Without Annealing Optimization	0.5%	5.48%	2 hr
With Annealing Optimization	5.48%	77.20%	23 min

For more information about annealing optimization, please refer to Appendix G of sphereface paper.

And also check out the discussion section of my report for a detailed description.

Exp3: The Impact of Class Number
During training, if we change the class number into the class number of APD dataset, then the network will be very hard to train. This happens only on the baseline model (sphereface).

Models listed below are trained with identical settings.
(10574 is the default class number of sphereface implementation, and 685 is the class number of APD dataset)

Model	Class Number	Performance (Training)
sphereface	10574	81.89%

Model	Class Number	Performance (Training)
sphereface	685	4.6%

Exp4: Overfitting?

Both sphereface_CELoss and sphereface_cos achieve more than 97% accuracy on training. However, this could be overfitting on training set. (closed-set inference is through id)

Model	Epoch Number	Performance (closed-set / open-set)
sphereface	20	74.8% / 79.6%
sphereface	29	74.8% / 79.6%
sphereface_CELoss	20	70.6% / 79.6%
sphereface_CELoss	29	70.7% / 79.6%
sphereface_cos	21	77.6% / 81.4%
sphereface_cos	60	76.6% / 75.9%
sphereface_cos	117	76.5% / 75.9%

Discussion

Annealing Optimization

The author of sphereface_pytorch implemented annealing optimization strategy. That is, in the beginning of optimization, we try to give the model a easier task (the original softmax loss is easier) and then a harder task (the modified loss is harder). Annealing optimization is done by the following equation:

$$f_{y_i} = \frac{\lambda \cos(\theta_{y_i}) + \phi(\theta_{y_i})}{1 + \lambda}$$

$\phi(\cdot)$ is the modified loss function.
By letting λ a large number (1500 in implementation), the loss function will approximately be the original version of softmax loss. Through the training process, let λ gradually decrease to a small number (5 in implementation) so that the loss function could be the modified version. $\phi(\cdot)$ is the modified loss function.

By this optimization technique, the training process could be more deterministic (without this technique, the performance of CosFace version on training set could vary from 71% to 78% through identical training process). Moreover, by this technique, the training process seems more reasonable: we have to split those faces into

clusters, and then apply the modified loss function to make interclass distance further while maintaining compact intraclass distance.

As for CosFace, my implementation is to directly replace the loss function after an iteration number with CosLoss. Also note that in the original setting, the model cannot converge really well (performance was lower than 80%, the baseline, even after 120 epochs which takes 2.5 hour to train). After adopting this optimization technique, CosFace model can achieve 80% accuracy in 13~14 epochs, which only takes less than 30 minutes to train.

Effect of Class Number

From the equation of loss functions, we can find that the main effect of reducing class number is on the size of W. Reducing the size of W may lessen the discriminative power of our model. However, it's only a hypothesis and is sort of weird (reduce the class number in CosFace version of our model doesn't lead to a bad result). Further investigation is needed to prove or disprove my hypothesis.

High Performance of CrossEntropyLoss Version on Training

I claim the high performance is due to the small size of APD dataset. Although sphereface_CELoss model has a better performance on training set, its testing performance is worse than even baseline model on closed-set. I'm not sure whether closed-set testing images are wholy chosen from training set (it seems a little weird if testing images are identical but the model performs poorly). Yet on open-set performance, three models have nearly the same performance. This shows the fact that these three models can somehow learn from faces. Though the performances are far from state-of-the-art models, their capabilities are not really bad, given the relative small dataset for training.

Overfitting

Except for sphereface_cos, the result of other two models shows that overfitting is slight. Furthermore, the result show that three models can converge in less than 30 epochs (can perform really well in 20 epochs) by annealing optimization technique.

Future Work

- Adopt ArcFace Loss into our model
  - ArcFace paper link
- Find out why does class number affect the performance of sphereface model
- Find out how class number affect our model
- Retrain sphereface model with the setting same with our CosFace version

References

- https://github.com/clcarwin/sphereface_pytorch
- https://arxiv.org/abs/1704.08063
- https://github.com/deepinsight/insightface/blob/fec2bf0b28c96aba9a4eacbae32fabdebd6bee94/recognition/arcface_torch/
- https://arxiv.org/abs/1801.07698
- https://zhuanlan.zhihu.com/p/76541084
- https://github.com/ipazc/mtcnn