CSE 252C - Advanced Computer Vision
HW 2

You-Yi Jau
Instructor: Manmohan Chandraker.
Collaborator: Sarah Wang, Tingwei Yu

May 25, 2019

1 Introduction

- In this report, we will study face recognition, covering face verification, face alignment and training of face recognition networks.

2 Problems and Results

1. **Using SphereFace [1] for Face Verification**

   - Run lfw eval.py and report the accuracy of SphereFace on LFW verification. **Shown in Table 1.**
   - Explain briefly how the following steps are performed when evaluating on LFW dataset:
     (a) Given the features extracted from the network, what distance metric is used to measure the distance between two faces? (lfw eval.py: Line 135)
        i. We use cosine distance.

```
cosdistance = f1.dot(f2)/(f1.norm()*f2.norm()+1e-5)
```

     (b) How to set up the threshold to determine whether two faces are from the same human? How to compute the accuracy? (lfw eval.py: Line 141 to 148)
        i. We split 6000 pairs of with positive and negative pairs into 10 folds. For each fold, we have 90% training and 10% testing data. We set the precision of the threshold to be 0.005. For each fold, we find the threshold with highest accuracy using training set and evaluate on testing set using the threshold. In the end, we take the mean and variance of the accuracy and the mean of the threshold.

```
accuracy = []
thd = []
folds = KFold(n=pairNum, n_folds=10, shuffle=False)
thresholds = np.arange(-1.0, 1.0, 0.005)
predicts = np.array([*map(lambda ...
    line:line.strip('\n').split(), predicts) ] )
for idx, (train, test) in enumerate(folds):
    best_thresh = find_best_threshold(thresholds, ...
        predicts[train] )
    accuracy.append(eval_acc(best_thresh, predicts[test]))
    thd.append(best_thresh)
print('LFWACC={:.4f} std={:.4f} ...
    thd={:.4f}'.format(np.mean(accuracy), ...
    np.std(accuracy), np.mean(thd)))
```

- An important step before face recognition is face alignment, in which we warp and crop the image based on the location of facial landmarks.
  (a) Briefly describe how we warp and crop the image. (lfw eval.py: Line 11-26)
    i. Given 5 pairs of points, we find an affine transformation for the pairs (line. 10). Then, we use the transformation matrix to warp and crop the image.

```
def alignment(src_img, src_pts):
    ref_pts = [ [30.2946, 51.6963],[65.5318, 51.5014],
        [48.0252, 71.7366],[33.5493, 92.3655],[62.7299, ...
            92.2041] ]
    crop_size = (96, 112)
    src_pts = np.array(src_pts).reshape(5,2)

    s = np.array(src_pts).astype(np.float32)
    r = np.array(ref_pts).astype(np.float32)

    tfm = get_similarity_transform_for_PIL(s, r)

    face_img = src_img.transform(crop_size, Image.AFFINE,
            tfm.reshape(6), resample=Image.BILINEAR)
    face_img = np.asarray(face_img )[:, :, ::-1]

    return face_img
```

  (b) Instead of doing face alignment, crop an image patch of height 112 pixels and width 96 pixels at the center of the image and report the accuracy.
    – The accuracy drops below 90%.
    – **Shown in Table 1.**

2. **Using MTCNN [2] for Detecting Face Landmarks**

- Run lfw landmark.py to generate the facial landmarks. Include two example outputs in your report.

Figure 1: Landmarks of the faces. (Right: No1, Left: No.10)

```
1 (No.1) AJ_Cook/AJ_Cook_0001.jpg 103.672 114.958 146.303 ...
     109.461 118.223 135.837 110.155 158.646 149.281 154.148
2
3 (No.10) Aaron_Pena/Aaron_Pena_0001.jpg  100.335 116.489 ...
     144.768 103.950 124.513 136.455 119.725 167.840 150.664 ...
     158.556
```

- Is the result better than using the landmarks provided in the previous question? If not, how can you improve performance?
  (a) No. The result is 0.7% worse than that from Sphereface itself **Shown in Table. 1**. We can fine-tine the model using the new landmarks to increase the results. The pretrained model is trained in another set of landmarks, but do inference based on the landmarks from MTCNN.
- What are the steps adopted by the method to achieve real-time speed?
  (a) They use cascaded structure and design lightweight CNN architecture for real time performance.
  (b) They apply less number but more discriminated of filter. "They reduce the number of filters and change the 5×5 filter to a 3×3 filter to reduce the computing while increasing the depth to get better performance."
  (c) They use simple online hard sample mining to improve the performance. They sort the samples by forward pass loss and select top 70% for training.
- Briefly describe how non-maximal suppression (NMS) is implemented in this method. (src/box utils.py: Line 5-68)
  (a) Given a bunch of boxes with probability, NMS helps take the ones with maximum probability over surrounding boxes.
  (b) (1) NMS sorts the boxes with probability. (2) select the box with largest probability (3) delete others s.t. IOU with the box larger than a threshold. (4) work on the queue (2)-(3) till all boxes are selected or deleted.

3. **Training SphereFace [1] on CASIA Dataset [3]**

- Implement the loss function and angle function [4].

- Train the network using casia train.py. You may try various hyperparameters like m, learning rate, batch size, training iterations and so on. Stop when you think the network behaves strangely (drop in accuracy, or loss stops decreasing). You may refer (and cite) any open source code. Include the following in your report:
  (a) Curves for training loss and accuracy on CASIA, ,which have been saved in the checkpoint directory (you may smooth the curves to make them look better).
    – I use m=4. Training accuracy rises up to 93%. **Shown in Figure 2** .
  (b) Accuracy on the LFW dataset, evaluated using lfw eval.py. You are expected to achieve accuracy higher than 90% on the LFW dataset.
    – 97.65% on LFW dataset. **Shown in Table 1.**
- The architecture above is a 20-layer residual network as described in Table 2 of [2], but without batch normalization. Now add batch normalization after every convolutional and fully connected layer. Train the new network on CASIA dataset and test on LFW dataset.
  – Draw the training curves for accuracy and loss on CASIA and compare to the curve without batch normalization.
    * Training accuracy is higher than that without batchnorm. **Shown in Figure 2** .
    * If we remove the batchnorm for fully connected layer. It becomes harder to train. We couldn't use the same learning rate.
  – Report accuracy on the LFW dataset, evaluated using lfw eval.py.
    * Higher accuracy (97.67%) on LFW dataset and lower std than that without batchnorm. **Shown in Table 1.**
  – Do you achieve better performance on LFW? If yes, explain how batch normalization helps. If not, try to explain why the results are worse.
    (a) Yes. Batch normalization help on the problem of internal covariate shift by training the mean and standard deviation for each batch.
    (b) For each mini-batch, we normalize the features across the same channel. Then, we use 2 parameters to control (expand) its mean and standard deviation.
    (c) The batch normalization helps the features to have better distribution for gradient to flow. It allows us to use higher learning rates.
- Randomly choose 10 identities from the CASIA dataset, forward pass all their images through the network and visualize the normalized features using tSNE [5] (Figure 4).

4. **Training CosFace [6] on CASIA Dataset [3]**

- Again implement the angle function and the loss function of CosFace under CustomLinear and CustomLoss in faceNet.py [7].

Figure 2: Training curve with smoothness rate 0.6. (Up: loss, Down: Accuracy), (checkpoint (red curve): sphereFace without Batch Norm(BN), add_bn_3 (blue): sphereFace with BN, cosf_sph20_2 (Magenta): cosFace with BN)

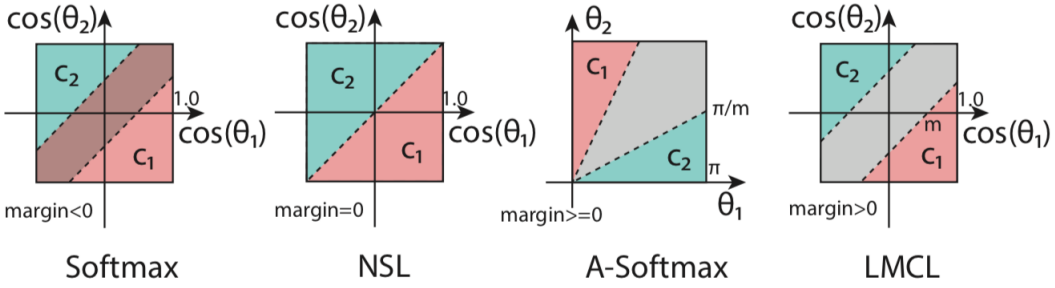

Figure 2. The comparison of decision margins for different loss functions the binary-classes scenarios. Dashed line represents decision boundary, and gray areas are decision margins.

Figure 3: Illustration of different decision margins. [6]

method	LFW Accuracy	std	thd
SphereFace	0.9918	0.0051	0.3095
SphereFace + crop center	0.8748	0.0221	0.2300
SphereFace + MTCNN	0.9845	0.0057	0.2995
Train SphereFace (m=4)	0.9765	0.0086	0.3970
Train SphereFace (add Batch Norm)	0.9767	0.0076	0.3720
Train CosFace (add Batch Norm)	0.9833	0.0058	0.2660

Table 1: Accuracy for different methods.

- Train the network on CASIA dataset. Again you are free to change any hyper parameters but report the hyper parameters that you think might influence the performance. Again draw the curve of loss and accuracy of training. Report the accuracy on LFW dataset.
  - I use m=0.35. Higher accuracy (98.33%) and lower std on LFW dataset than the result from sphereFace. **Shown in Table 1.**
  - Training curve. **Shown in Figure 2** .
- If you achieve better performance compared to SphereFace, well done! Can you provide a reason? If you do not outperform SphereFace, can you provide a cause?
  - Yes. It is slightly better than the accuracy from sphereFace. The main reason is that the loss function defines a clearer and larger decision boundary for the features (Figure. 3).
- Plot the tSNE visualization of the CosFace embedding for the same identities from the CASIA dataset used to visualize the SphereFace embedding in the previous question (Figure 4).

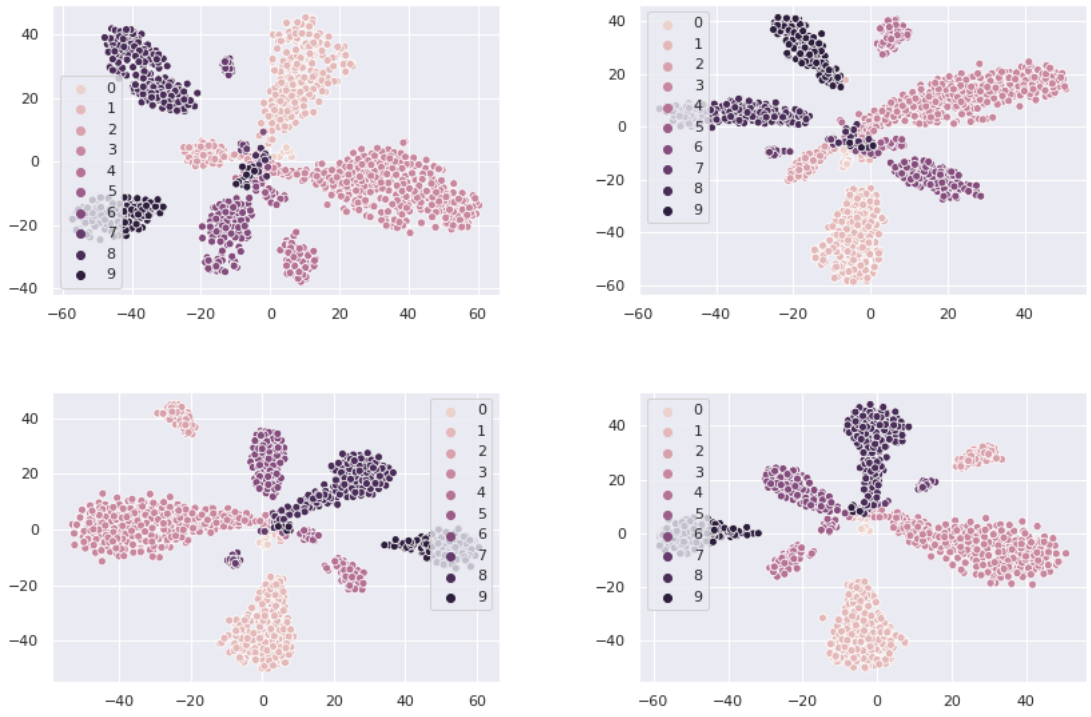

Figure 4: tSNE from different models. (Upper left: pretrained model from [4], Upper right: My model trained from [4], lower left: My model with batch normalization from [4], lower right: My model of implementing cosFace [7].)

References

[1] W. Liu, Y. Wen, Z. Yu, M. Li, B. Raj, and L. Song, "SphereFace: Deep hypersphere embedding for face recognition." [Online]. Available: http://arxiv.org/abs/1704.08063

[2] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao, "Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks," *IEEE Signal Processing Letters*, vol. 23, no. 10, pp. 1499–1503, Oct. 2016, arXiv: 1604.02878. [Online]. Available: http://arxiv.org/abs/1604.02878

[3] D. Yi, Z. Lei, S. Liao, and S. Z. Li, "Learning Face Representation from Scratch," *arXiv:1411.7923 [cs]*, Nov. 2014, arXiv: 1411.7923. [Online]. Available: http://arxiv.org/abs/1411.7923

[4] Clcarwin, "clcarwin/sphereface_pytorch," Oct 2017. [Online]. Available: https://github.com/clcarwin/sphereface_pytorch

[5] L. v. d. Maaten and G. Hinton, "Visualizing Data using t-SNE," *Journal of Machine Learning Research*, vol. 9, no. Nov, pp. 2579–2605, 2008. [Online]. Available: http://www.jmlr.org/papers/v9/vandermaaten08a.html

[6] H. Wang, Y. Wang, Z. Zhou, X. Ji, D. Gong, J. Zhou, Z. Li, and W. Liu, "CosFace: Large margin cosine loss for deep face recognition." [Online]. Available: http://arxiv.org/abs/1801.09414

[7] MuggleWang, "Mugglewang/cosface_pytorch," Oct 2018. [Online]. Available: https://github.com/MuggleWang/CosFace_pytorch

3 Code

Please refer to 'faceNet.py' under sphereFace and cosFace for the implementation details.