Homework 2 Part 2

Face Verification using Convolutional Neural Networks

11-785: Introduction to Deep Learning (FALL 2020)

DUE: **10/18/2020 11:59 PM ET**

1 Introduction

Face recognition can be categorized into face classification and face verification. Given an image of a person's face, the task of classifying the ID of the face is known as face classification, which is a closed-set problem. The task of determining whether two face images are of the same person is known as face verification, which is an open-set problem[1].

In this assignment, you will use Convolutional Neural Networks (CNNs) to design an end-to-end system for **face verification**. Your system will be given two images as input and will output a score that quantifies the similarity between the *faces* in these images. This helps us decide whether the faces from the two images are of the same person or not.

You will train your model on a dataset with a few thousand images of labelled ID's (i.e., a set of images, each labeled by an ID that uniquely identifies the person). You will learn more about embeddings (in this case, embeddings for face information), several loss functions, and, of course, convolutional layers as effective shift-invariant feature extractors. You will also develop skills necessary for processing and training neural networks with big data, which is often the scale at which deep neural networks demonstrate excellent performance in practice.

2 Face Verification

The input to your system will be a *trial*, i.e., a pair of face images that may or may not belong to the same person. Given a trial, your goal is to output a numeric score that quantifies how similar the faces in the two images are. One straightforward approach is to flatten each image matrix into a vector and then to compute the Euclidean distance between two vectors. A lower distance will indicate higher confidence that the faces in the two images are of the same person. If you get a competitive result on Kaggle by applying this approach(you can also define your own distance metric functions), you can skip the following texts except for *Dataset* and *Submission* sections because you have finished hw2p2. If you do not get a desirable result or if you don't want to hurt your CNN's feelings, the following instructions might help you out.

2.1 Face Embedding

We might not really encourage directly computing the distance between two image matrices for two reasons. First, flattened image vectors are typically high-dimensional, which results in additional computation costs. Second, original image features are not discriminative enough. Your task in this assignment is to train a CNN model to extract a compact, low-dimensional feature, which keeps the most important information of the image and is also discriminative. This compact feature will be represented in a *fixed-length* vector, known as a **face embedding**. Given two face embeddings, you will use an appropriate metric between the embeddings to produce your similarity scores. Tips on how to choose a proper distance metric will be covered later.

[1]For close-set task, all testing identities are predefined in training set. For open-set task, testing identities typically do not appear in training set

2.2 Getting Started

If you have trained your CNN, your end-to-end face verification system will use your CNN as follows - given two images, each image will be passed through the CNN to generate corresponding face embeddings, between which you will compute your similarity score. Your system will output this score. The next question is: **how should you train your CNN to produce high-quality face embeddings?**

There are typically two popular approaches, both of which are able to give SOTA results. Feel free to choose one of them or the combination of these two.

2.2.1 N-way Classification

Classification is a good start. Similar to speech classification in the previous assignment, you are able to apply CNNs for face classification. Suppose the labeled dataset contains a total of M images that belong to N different people (here, $M > N$). Your goal is to train your model on this data so that it produces "good" face embeddings. You can do this by optimizing these embeddings for predicting the face IDs from the images. More concretely, your network will consist of several (convolutional) layers for feature extraction. The input will be (possibly a part of) the image of the face. The output of the *last* such feature extraction layer is the face embedding. You will pass this face embedding through a linear layer with dimensions `embedding_dim` $\times$ `num_faceids`, followed by softmax, to classify the image among the N (i.e., `num_faceids`) people. You can then use cross-entropy loss to optimize your network to predict the correct person for every training image. After the network is trained, you will remove the linear/classification layer. This leaves you with a CNN that computes face embeddings given arbitrary face images.

A high testing classification accuracy will **probably** indicate that your feature extractor is good enough to generate discriminative face embeddings. You are encouraged to explore the interconnection between classification accuracy and verification performance.

Though a good job in classification is guaranteed to help you reach the A-cutoff, you are free to apply advanced loss functions such as Center-loss [1], LM [2], L-GM [3], SphereFace [4], CosFace [5], ArcFace [6], UniformFace [7] to go beyond this.

2.2.2 Metric Learning

The second approach is actually called deep metric learning(DML): Instead of modeling the classes, you are directly modeling the similarity between two images. The general goal is to make the minimum distance between negative pairs larger than the maximum distance between positive pairs[2].

A potential approach is to build a Siamese Neural Network [8] and apply a Contrastive loss function as follows:

$$L = \frac{1}{N}\Sigma_{i=1}^{N}[y * d(P_i) + (1-y) * (m - d(P_i))] \tag{1}$$

Where d denotes Euclidean distance, and $y = 1/0$ indicates the pair P_i is positive/negative respectively. m is a margin. N denotes total number of training objectives.

There are two popular approaches to make pairs for your verification system. One is **offline selection**: pairs are generated before passed through the neural network. Another is **online selection**: pairs are generated in the mini-batch during training. For offline selection, please pay attention to the ratio of #negative pairs to #positive pairs. You are **advised** to set this ratio as 5:5, 6:4, 7:3. For online selection, one straightforward method is to select all $\frac{B(B-1)}{2}$ pairs within a mini-batch of size B. You can also just select *hard*[3] pairs within the mini-batch, which is also referred to as **Hard Sample Mining** [9, 10].

Instead of measuring the similarity between pairs, you can also apply Triplet loss [11] or Quadruplet loss [12] to model the similarities among triplets or quadruplets.

[2]Two instances in the positive pair should be from the same identity. Two instances in the negative pair should be from different identities.

[3]Large similarity for negative pairs and small similarity for positive pairs

If you're wondering if there exists a Quintuplets, Sextuplets, Septuplets or even Octuplets loss, you can refer to the N-pair Loss [13], Lifted-Structure Loss [14], Softtriplet Loss [15] papers.

It may also be possible for other advanced loss functions such as Pair-Wise Loss [16], Multi-Similarity(MS) [17], Mask Proxy(MP) [18] to give SOTA verification performance.

2.3 Loading Training Data

For loading the images, we recommend that you look into the ImageFolder dataset class of PyTorch at `https://pytorch.org/docs/stable/torchvision/datasets.html#imagefolder`. The images in subfolders of `classification_data` are arranged in a way that is compatible with this dataset class. Note that `ImageFolder` is helpful for both `N-way classification`, and `Metric Learning` tasks.

2.4 System Evaluation

This subsection briefly describes how the "quality" of your similarity scores will be evaluated. Given similarity scores for many trials, some *threshold* score is needed to actually accept or reject pairs as *same-person* pairs (i.e., when the similarity score is above the threshold) or *different-person* pairs (i.e., when the score is below the threshold), respectively. For any given threshold, there are four conditions on the results: some percentage of the different-person pairs will be accepted (known as the *false positive* rate), some percentage of the same-person pairs will be rejected (known as the *false rejection* rate), some percentage of the different-person pairs will be rejected (known as the *true negative* rate), and some percentage of the same-person pairs will be accepted (known as the *true positive* rate).

The Receiver Operating Characteristic (ROC) curve is created by plotting the True Positive Rate (TPR) against the False Positive Rate (FPR) at various threshold settings [4]. The Area Under the Curve (AUC) for the ROC curve is equal to the probability that a classifier will rank a randomly chosen similar pair (images of same people) higher than a randomly chosen dissimilar one (images from two different people) (assuming 'similar' ranks higher than 'dissimilar' in terms of similarity scores).

This is the metric which will be used to evaluate the performance of your model for the face verification task.

To track your progress, after an epoch of training, you can compute a similarity score for every trial in the validation set, write them to another file. One suggested approach to compute AUC is to use the function provided in sklearn library[5]:

- `sklearn.metrics.roc_auc_score(true_label, similarity_scores)`. This function is useful for Verification Validation. It loads the true label array and the generated similarity scores array and prints out the average AUC score. Please also pay attention to the difference between cosine similarity score and Euclidean distance score.

2.5 Cosine Similarity VS Euclidean Distance

You may struggle with selecting a proper distance metric for the verification task. The most two popular distance metrics used in verification are cosine similarity and Euclidean distance. We would tell you in that both two metrics are able to reach SOTA score, but at least you should get an intuition on how to choose one of them.

The metric should be training-objective-specific, where training objective refers to the loss function. Let us start with revisiting softmax cross entropy:

$$Loss = -\frac{1}{N}\Sigma_{i=1}^{N} log \frac{e^{W_{Y_i}^T X_i}}{\Sigma_{j=1}^{N} e^{W_{Y_j}^T X_i}} \tag{2}$$

[4] `https://en.wikipedia.org/wiki/Receiver_operating_characteristic`
[5] `https://scikit-learn.org/stable/`

Where Y_i is the label of X_i. If you take a thorough look at this formula, you will find that the objective is to make the vector(embedding) X_i be closer to the vector W_{Y_i} and be far away from other vectors W_{Y_j}. Under this rule, the W_{Y_i} is actually the center of $i-th$ class. Because you are performing dot product between the class center and the embedding, then each embedding would be similar to its center in the **Angular Space**, which could be illustrated in the following Figure. 1. So during verification, you are strongly suggested to apply cosine similarity rather than Euclidean distance to compute the similarity score.

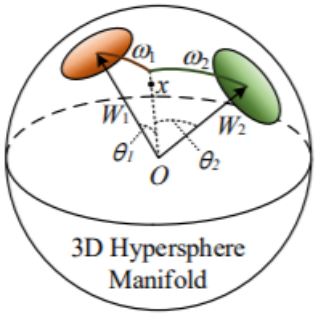

Figure 1: Angular Space [4]

Furthermore, if we design our own loss function e.g. in Eq. 3, you are suggested to apply Euclidean distance metric to compute similarity. (Is this RBF?)

$$Loss = -\frac{1}{N}\Sigma_{i=1}^{N} log \frac{e^{||W_{Y_i} - X_i||^2}}{\Sigma_{j=1}^{N} e^{||W_{Y_j} - X_i||^2}} \tag{3}$$

Question left to you, what metric is **probably** better if you start with metric learning and apply the loss function in Eq. 1?

However, the aforementioned conclusions are not definitely true. We would tell you that sometimes Euclidean distance is also good when you apply softmax XE in Eq. 2 and cosine similarity is also good when you apply Eq. 3 as loss function. We would just give you the following hint and let you explore it.

$$||U - V||_2^2 = ||U||_2^2 + ||V||_2^2 - 2U^T V \tag{4}$$

3 Dataset

The data for the assignment can be downloaded from the Kaggle competition link [6]. The dataset contains images of size 64×64 for all RGB channels.

3.1 File Structure

The structure of the dataset folder is as follows:

- `classification_data`: Each sub-folder in `train_data`, `val_data` and `test_data` contains images of one person and the name of that sub-folder represents their ID.
  - `train_data`: You are supposed to use the `train_data` set for training your model **either for the classification task or for the metric learning task.**
  - `val_data`: If you are doing with classification task, you are supposed to use `val_data` to validate the classification accuracy. If you are doing with metric learning task, you can skip this folder.

[6] `https://www.kaggle.com/c/11-785-fall-20-homework-2-part-2/overview`

  - `test_data`: If you are doing with classification task, you are supposed to use `test_data` to test the classification accuracy. If you are doing with metric learning task, you can skip this folder.

- `verification_data`: This is the directory that contains the images for both the `Verification Validation` and `Verification Test`.
- `verification_pairs_val.txt`: This file contains the trials for `Verification Validation`. The first two column are the images path of the trial. The third column contains the true label for the pair. You are supposed to use the data in this file to validate your AUC score.
- `verification_pairs_test.txt`:This file contains the trials for `Verification Test`. The first two column are the images path of the trial. You task is to compute the similarity between each two trials and to generate submission file based on this.
- `hw2p2_sample_submission.csv`: This is a sample submission file.

4 Submission

Following are the deliverables for this assignment:

- Kaggle submission for Face Verification[7].
- A one page write up (**PDF**) describing your model architecture, loss function, hyper parameters, any other interesting detail which led to your best result. Please limit the write up to one page. The link for submitting the writeup will be posted later on Piazza.

5 Conclusion

That's all. As always, feel free to ask on Piazza if you have any questions.

Good luck and enjoy the challenge!

References

[1] Yandong Wen, Kaipeng Zhang, Zhifeng Li, and Yu Qiao. A discriminative feature learning approach for deep face recognition. In European conference on computer vision, pages 499–515. Springer, 2016.

[2] Weiyang Liu, Yandong Wen, Zhiding Yu, and Meng Yang. Large-margin softmax loss for convolutional neural networks. ProC. Int. Conf. Mach. Learn., 12 2016.

[3] W. Wan, Y. Zhong, T. Li, and J. Chen. Rethinking feature distribution for loss functions in image classification. In 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 9117–9126, 2018.

[4] Weiyang Liu, Yandong Wen, Zhiding Yu, Ming Li, Bhiksha Raj, and Le Song. Sphereface: Deep hypersphere embedding for face recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 212–220, 2017.

[5] H. Wang, Yitong Wang, Z. Zhou, Xing Ji, Zhifeng Li, Dihong Gong, Jingchao Zhou, and Wenyu Liu. Cosface: Large margin cosine loss for deep face recognition. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 5265–5274, 2018.

[6] Jiankang Deng, J. Guo, and S. Zafeiriou. Arcface: Additive angular margin loss for deep face recognition. 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 4685–4694, 2019.

[7] `https://www.kaggle.com/c/11-785-fall-20-homework-2-part-2/overview`

[7] Y. Duan, J. Lu, and J. Zhou. Uniformface: Learning deep equidistributed representation for face recognition. In 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 3410–3419, 2019.

[8] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks for one-shot image recognition. 2015.

[9] Weifeng Ge, Weilin Huang, Dengke Dong, and Matthew R. Scott. Deep metric learning with hierarchical triplet loss. In ECCV, 2018.

[10] R. Manmatha, Chao-Yuan Wu, Alexander J. Smola, and Philipp Krähenbühl. Sampling matters in deep embedding learning. 2017 IEEE International Conference on Computer Vision (ICCV), pages 2859–2867, 2017.

[11] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 815–823, 2015.

[12] Weihua Chen, Xiaotang Chen, Jianguo Zhang, and Kaiqi Huang. Beyond triplet loss: a deep quadruplet network for person re-identification. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 403–412, 2017.

[13] Kihyuk Sohn. Improved deep metric learning with multi-class n-pair loss objective. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, Advances in Neural Information Processing Systems 29, pages 1857–1865. Curran Associates, Inc., 2016.

[14] Hyun Oh Song, Yu Xiang, Stefanie Jegelka, and Silvio Savarese. Deep metric learning via lifted structured feature embedding, 2015.

[15] Qi Qian, Lei Shang, Baigui Sun, Juhua Hu, Hao Li, and Rong Jin. Softtriple loss: Deep metric learning without triplet sampling, 2019.

[16] H. Dhamyal, T. Zhou, B. Raj, and R. Singh. Optimizing neural network embeddings using a pairwise loss for text-independent speaker verification. In 2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU), pages 742–748, 2019.

[17] Xun Wang, Xintong Han, Weiling Huang, Dengke Dong, and Matthew R. Scott. Multi-similarity loss with general pair weighting for deep metric learning. 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 5017–5025, 2019.

[18] Mask proxy loss for text-independent speaker recognition. `https://drive.google.com/file/d/1XQ2vLhQWnRXUfiS-JR_g9m_taNVS11fR/view?usp=sharing`, 2020.