Homework 2 Part 2

Face Verification via Convolutional Neural Networks

11-785: Introduction to Deep Learning (Fall 2019)

OUT: **September 30, 2019**
DUE: **October 20, 2019**

1 Introduction

Given an image of a person's face, the task of classifying the ID of the face is known as **face classification**. Whereas the problem of determining whether two face images are of the same person is known as **face verification** and this has several important applications.

In this assignment, you will use Convolutional Neural Networks (CNNs) to design an end-to-end system for face classification and face verification. For the classification task, your system will be given an image of a face as input and will output the ID class of the face. As for the verification task, your system will be given two images as input and will output a score that quantifies the similarity it detects among the *faces* in these images. Ultimately, this allows us to decide whether the faces from the two images are of the same person or not.

To this end, you will train 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). In doing so, you will gain experience with the concept of *embeddings* (in this case, embeddings for face information), optionally several relevant loss functions, and, of course, convolutional layers as effective shift-invariant feature extractors. Additionally, you will begin to develop skills necessary for processing and training with *big data*, which is often the scale at which deep neural networks demonstrate excellent performance in practice.

2 Face Classification

An input to your system is a face image and you have to predict the ID of the face. The true face image ID will be present in the training data and so the network will be doing an N-way classification to get the prediction. You are provided with the development set and can fine tune the model based on the accuracy you get on the development set.

3 Face Verification

An input to your system is a *trial*, that is, 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 of the two images appear to be. On some scale, a higher score will indicate higher confidence that the faces in the two images are of the same person.

Below, we provide a brief outline of *how* you might use CNNs to compute such similarity scores effectively for this task.

3.1 Face Embeddings

Intuitively, facial features vary extensively across people. Your main task is to train a CNN model to extract and represent such important features from a person. These extracted features will be represented in a *fixed-length* vector of features, known as a **face embedding**. Given two face embeddings, you will use an appropriate metric (e.g., *cosine similarity* or (negative) *L2 distance* between the computed embeddings) to produce your similarity scores. We recommend you use (or at least begin by using) cosine similarity.

Accordingly, once 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, among which you will compute your similarity score. This score will be the output of your overall system. The next natural question is: how should you train your CNN to produce high-quality face embeddings?

3.2 N-way Classification

This is the simplest approach and the one we will (mostly) focus on in this assignment. The Face Classification part will naturally lead you into this solution.

Let our labeled dataset contain M images of N unique people (here, $M > N$). Your goal is to use this data to train your CNN so that it produces "good" face embeddings. One reasonable way to do so is to optimize these embeddings for predicting the identities of the training face from their images. The resulting embeddings will clearly encode a lot of face-varying features, just as desired. This suggests an N-way classification task, where you classify every image among N classes (i.e., one for every unique person in the training set).

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 images 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.

3.3 Alternative Approaches

Once you train your N-way classifier, you can either (a) continue training the classifier with loss functions such as contrastive-loss [1], center-loss [2], angular-softmax [1], etc. Alternatively, (b) you can remove the layer entirely and optimize the net using comparator-losses that optimize verification, e.g. triplet-loss[3], pair wise loss [4]. Other interesting papers to look at are: [5].

3.4 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 [1]. 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 access the performance of your model for the face verification task.

4 Dataset

The data for the assignment can be downloaded from the :

`https://cmu.box.com/s/daslpynscr6xw9lp89eek8l3326dv8ii`

A kind reminder here, aws update their instance environments. Right now an instance by default has 75 gb memory, but it contains 65 gb environemnt files. You will have to increase the storage size when you lauch the aws insance. 100 gb will be enough for the purpose of this assignment.

The data contains images of size 32 by 32. The images have a pinkish shade to them, that is because they have been transformed, meaning scaled and rotated. This has been done so that no external pre-trained models can be used on the dataset.

This assignment contains 2 parts. One is face classification, and the other is face verification. For classification, you will be given a human face image. What you need to do is to learn to classify this image into correct people IDs. For verification, you will be given two images, and need to calculate the similarity score for those images using the embeddings generated by your classification network.

It's very important to note that, for classification, the train, validation and test contains the same set of images. what it means is that, your network cannot classify images unless it has seen it before, and the dataset is set up that way.

For verification test, the images are from different sets. For the test - your network should be able to tell whether two images belong to the same person or not, even if it has never seen those people before.

4.1 File Structure

In total, we have 8 files for this assignment, including 3 '.txt' files and 5 '.tar' files. Their functions and structure of organization are explained below.

- `train_data.tar`: This file contain two sub files. One is called medium, the other is called large. You are supposed to train for classification using medium set. The folder number is people's ID, which is your training label.
- `validation_classification.tar`: This file too contains medium and large, similar to the medium and large in the train_data.tar.
- `test_classification.tar`: this file contains the test data for the classification task. You will have to use the order specified by the `test_order_classification.txt` to load the test images and to generate the labels for them respectively. The order will be used for kaggle. Note that the photo ID right now is not the same as the face ID in the previous files.
- `test_order_classification.txt`: this file specifies the order your should read the previous image file (`test_classification.tar`). You should generate your predictions following the same order and submit it to the Kaggle Leaderboard.

[1] `https://en.wikipedia.org/wiki/Receiver_operating_characteristic`

Above are the files you need to use to do the classification task. You don't have to use the large dataset to do the classification, but you can use it to further fine tune your model. The classification leaderboard on Kaggle will only be tested on the medium dataset. People might ask, why we have the large dataset then. Well, once you are done with classification, we want you to use both the medium and large dataset to fine tune your model for verification. Keep in mind that you are still doing the classification task at this phase. You will have to merge the validation-medium and validation-large to get a new validation set (let's call this new validation set vali-new) to ensure that your model does not over fit during this phase.

- `validation_verification.tar`: this is the file you use to validate your verification task model. The `validation_verification` is for verification alone, it contains the photos that your model has not seen before. The `validation_trials_verification.txt` are the trials created from this dataset and you need to calculate the AUC score for this trial file to get an idea how your model is performing on verification. You can use the utils.py file to calculate the AUC score.
- `validation_trials_verification.txt`: This file contains the trials for the `validation_verification`. The first two column are the images path of the trial. The third column contains the true label for the pair. 1 meaning they are images of the same person, 0 meaning they are images of different people.
- `test_verification.tar`: this is the file that contains the images for the verification test. The `test_trials_verification_student.txt` contains the trials created from this dataset.
- `test_trials_verification_student.txt`: this file provides the trials for the `test_verification.tar`. This is the file you submit on kaggle. Make sure to keep the order the same as the order specified in this file. Add another column to specify the score of the match. Look at kaggle for more explanation.

5 Getting Started

Below, we describe how you should (*i*) load subsets of the training data while prototyping and experimenting, (*ii*) train with N-way classification on the 32x32 images, and (*iii*) evaluate your AUC on the validation set. The starter code uses the following pip packages: `sklearn`, `PIL`, `torchvision`.

5.1 Preprocessing

In this homework, there is not much of pre-processing to be done on the images. The following two steps mentioned are optional. These steps are used in such areas and related tasks in the research domain. It may give you a boost in performance of the model.

Face Detection: Face detection is the automatic process for detection of human faces in digital images. This will ensure that the model you are training only sees images of humans and any noise in the images is deleted.

Face Alignment: Face alignment is the automatic process of identifying the geometric structure of human faces in digital images. Given the location and size of a face, it automatically determines the shape of the face components such as eyes and nose [2]. Given the location of the face in the image, images can be cropped to include only the human faces, without any background noise. This will also reduce noise for the model training.

5.2 Loading Training Data

For Loading the images, we recommend you look into the ImageFolder DataSet of PyTorch at `https://pytorch.org/docs/stable/torchvision/datasets.html#imagefolder`. This assumes that the training

[2] `https://link.springer.com/referenceworkentry/10.1007%2F978-0-387-73003-5_186`

images from a specific ID are arranged in one folder, which is exactly how the training data and the validation data provided to you are arranged. However, you would need to think how to handle the test data for your problem statement.

5.3 Validation with AUC

- `python score.py <score csv file> <label csv file>`: This file is useful for validation trials. It would load the true score csv and the generated scores csv, the file will print out the AUC calculated.

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 and then use the `AUC` function provided.

6 Submission

You will be submitting three things in this part of the homework:

- Kaggle submission for Face Classification.
- Kaggle submission for Face Verification.
- A one page write up describing what model architecture, loss function, hyper parameters, any other interesting detail led to your best result for the above two competitions. [Please limit the write up to one page. It will make it easier for us to check it. The way to submit the write up will be posted later on piazza].

7 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] 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.

[2] 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.

[3] 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.

[4] Optimizing neural network embeddings using pair-wise loss for text-independent speaker verifiation. `https://web2.qatar.cmu.edu/~hyd/pair_wise_ppr.pdf`.

[5] 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.