Homework 2 Part 2

Face Classification & Verification using Convolutional Neural Networks

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

DUE: **03/08/2020 11:59 PM ET**

1 Introduction

Given an image of a person's face, the task of classifying the ID of the face is known as **face classification**. 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 these tasks. For the classification task, your system will be given an image of a face as input and will output the ID of the face. For the verification task, 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 Classification

The input to your system will be a face image and you will have to predict the ID of the face. The ground truth 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 a validation set for fine-tuning your model.

3 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. 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) L_2 *distance* between the embeddings) to produce your similarity scores. We recommend that you use (or at least begin with) cosine similarity.

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,

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?

3.2 N-way 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. The resulting embeddings will encode a lot of discriminative facial features, just as desired. This suggests an N-way classification task.

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.

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 the network for the verification task, 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 kaggle competition link [2]. The dataset contains images of size 32×32.

For classification, all the identities are predefined in the training set. This is known as closed-set protocol.

[1] `https://en.wikipedia.org/wiki/Receiver_operating_characteristic`
[2] `https://www.kaggle.com/c/11-785-s20-hw2p2-classification/data`

For verification, the test identities are disjoint from the training identities, i.e. 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. This is known as open-set protocol.

4.1 File Structure

The structure of the dataset folder is as follows:

- `train_data.tar`: This file contains two sub folders (`medium` and `large`). You are supposed to use the `medium` set for training your model for the classification task. Each sub-folder in `medium` contains images of one person and the name of that sub-folder represents their ID.
- `validation_classification.tar`: This file has the same structure as `train_data.tar`.
- `test_classification.tar`: This file contains the test data for the classification task. You will have to use the order specified in `test_order_classification.txt` to load the test images and to predict their labels. The same order should be used in your kaggle submission.
- `test_order_classification.txt`: This file specifies the order in which you should generate your predictions for the Kaggle leaderboard.

  Note: You don't have to use the large dataset for the classification task, but you can use it to fine tune your model for the verification task. You might want to merge validation-medium and validation-large to get a new validation set 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 from identities 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 of how your model is performing on verification. You can use the `utils.py` (`https://www.kaggle.com/c/11-785-s20-hw2p2-verification/overview/evaluation`) file to calculate the AUC score.
- `validation_trials_verification.txt`: This file contains the trials for `validation_verification`. The first two column are the images path of the trial. The third column contains the true label for the pair.
- `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 specifies the order of the trials for `test_verification.tar`.

5 Getting Started

5.1 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 the training and validation folders are arranged in a way that is compatible with this dataset class. You will need to handle the test data using your own custom dataset class.

5.2 Validation with AUC

- `python utils.py <score csv file> <label csv file>`: This python script is useful for validation trials. It loads the true score csv and the generated scores csv and prints out the average AUC score.

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

Following are the deliverables for this assignment:

- Kaggle submission for Face Classification.
- Kaggle submission for Face Verification.
- A one page write up describing your 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. The link for submitting the writeup 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.