ICS 471, Term 201
Artificial Neural Networks and Deep Learning

HW# 4

Due date: Sunday, Nov. 29, 2020

Face Verification using Convolutional Neural Networks

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).

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. This approach is not encouraged 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.

2.1 Face Embedding

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.

[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?**

N-way Classification

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×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.

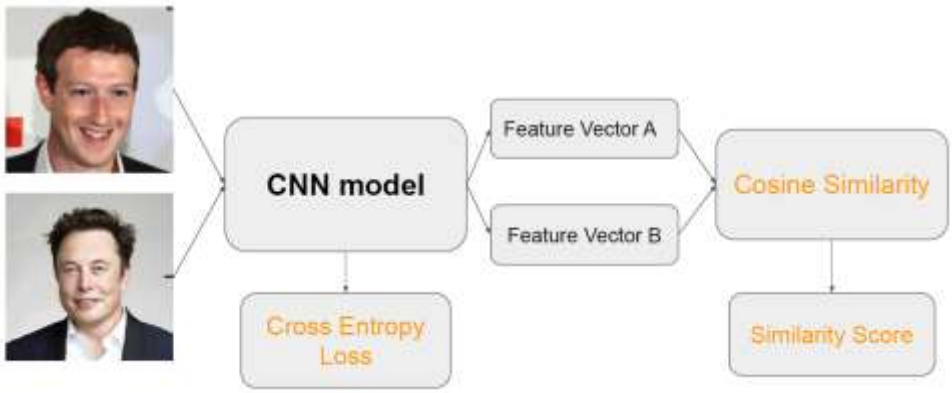
Cosine Similarity

One of the most popular distance metrics used in face verification is cosine similarity. Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space. It is defined to be equal to the cosine of the angle between them, which is also the same as the inner product of the same vectors normalized to both having length of 1.

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

You can find out cosine similarity implementation in PyTorch here.

The following diagram shows the flow of your solution:



System Evaluation

This subsection briefly describes how the \quality" of your similarity scores can 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 negative 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 [2]. 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 that you should use 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[3]:

- 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.

3. Dataset

The data for the assignment can be downloaded from here. The dataset contains images of size 64 × 64 for all RGB channels.

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 (4000 directories)
  - **tal_data**: You are supposed to use the val data to validate the classification accuracy (4000 directories)
  - **test_data**: You are supposed to use test data to test the classification accuracy (4000 directories)
- **verification_data**: This is the directory that contains the images for the Verification Validation
- **Verification_pairs_val.txt**: This file contains the trials for Verification Validation. The first two columns are the images path of the trial. The third column contains the true label for the pair.

4. Submission

You need submit a report describing your solution and the results obtained in terms of your CNN architecture training accuracy, validation accuracy and test accuracy. In addition, you should report the accuracy of your similarity prediction based on the data given in **verification_pairs_val.txt.** You also need to submit the notebook which has your code.

[2] https://en.wikipedia.org/wiki/Receiver_operating_characteristic
[3] https://scikit-learn.org/stable/