Assignment 1 - Kaggle Part

Face Classification using Convolutional Neural Network

AIST 4010: Foundation of Applied Deep Learning (Spring 2024)

DUE: **11:59PM (HKT), Feb. 26, 2024**

1 Introduction

Even though face recognition may sound quite trivial to humans, it has remained a challenging computer vision problem in the past decades. Thanks to deep learning methods, computers now can leverage huge datasets of faces to learn rich and compact representations of human faces, allowing models even to outperform the face recognition capabilities of humans.

Face recognition mainly consists of two parts. 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**, an open-set problem[1].

In this assignment, you will use Convolutional Neural Networks (CNNs) to design an end-to-end system for the face classification task (well, other techniques are required if you want to get higher scores.) In this task, your system will be given an image of a face as input and should output the ID of the face.

You will train your model on a dataset with a few thousand images of labeled IDs (i.e., a set of images, each labeled by an ID that uniquely identifies the person.) You will learn more about embeddings[2], 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.

- Goal: Given a person's face, return the ID of the face
- Kaggle: `https://www.kaggle.com/c/aist4010-spring2024-a1`

2 Face Classification

2.1 Face embedding

Before we dive into implementation, let's ask ourselves a question: how do we differentiate faces? Yes, your answers may contain skin tone, eye shapes, etc. Well, these are called facial features. Intuitively, facial features vary extensively across people (and make you different from others). Your main task in this

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

[2] In this case, embeddings for face information.

assignment is to train a CNN model to extract and represent such important features from a person (You can also include some additional architectures or techniques to improve your performance). These extracted features will be represented in a fixed-length vector of features, known as a **face embedding**.

Once your model can encode sufficient discriminative facial features into face embeddings, you can pass the face embedding to a fully-connected(FC) layer to generate the corresponding ID of the given face.

Now comes our second question: how should we train your CNN to produce high-quality face embeddings?

2.2 Multi-class Classification

It may sound fancy, but conducting face classification is just doing a multi-class classification: the input to your system is a face image, and your model needs to predict the ID of the face.

Suppose the labeled dataset contains a total of M images that belong to N different people (where $M > N$). Your goal is to train your model on this dataset so that it can produce "good" face embeddings. You can do this by optimizing these embeddings to predict the face IDs from the images. The resulting embeddings will encode many discriminative facial features, just as desired. This suggests an N-class classification task.

A typical multi-class classifier conforms to the following architecture:

$$\text{Classic multi-class classifier} = \text{feature extractor(CNN)} + \text{classifier(FC)}$$

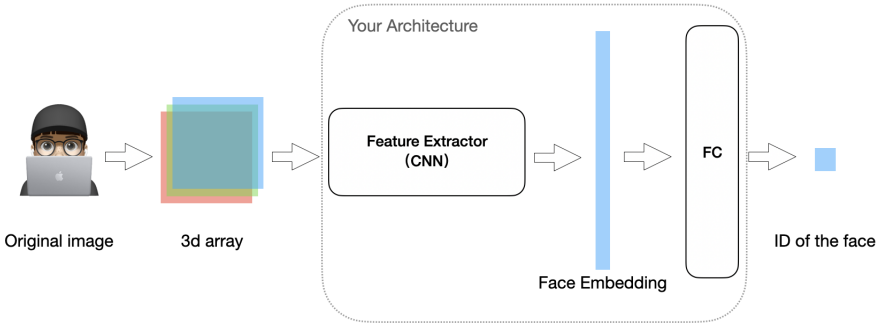

Figure 1: A typical face classification architecture

More concretely, your network consists of several (convolutional) layers for feature extraction. The input will be (possibly a part[3] 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 whose dimension is *embedding dim* $\times$ *num of faceids*, followed by Softmax, to classify the image among the N (i.e., num of faceids) people. You can then use cross-entropy loss to optimize your network to predict the correct person for every training image.

The ground truth will be provided in the training data (making it supervised learning). You are also given a validation set for fine-tuning your model. Please refer to the **Dataset section** where you can find more details about what dataset you are given and how it is organized. To understand how we (and you) evaluate your system, please refer to the **System Evaluation section**.

That's pretty much everything you need to know for your first Kaggle competition. Go for it!

[3]It depends on whether you pre-process your input images

3 Dataset

The data for the assignment can be downloaded from the Kaggle competition link[4]. The dataset contains images of size 64×64 for all xyz channels. In this competition, we are dealing with faces from 1000 people (That being said, we have 1000 classes.)

You will be given a human face image. What you need to do is to learn to classify this image into correct people IDs.

3.1 File Structure

The structure of the dataset folder is as follows:

- `classification-data`: Each sub-folder in train, val and test contains images of one person, and the name of that sub-folder represents their ID.
  - `train`: You are supposed to use the train data set to train your model for the classification task.
  - `val`: You are supposed to use val data to validate the classification accuracy.
  - `test`: You are supposed to assign IDs for images in test data and submit your result.
- `test_list.txt`: This file contains the file names of the test set. Your task is to assign an ID to each image and generate a submission file based on the order given here.
- `sample_submission.csv`: This is a sample submission file for this competition.

3.2 Loading Training Data - ImageFolder

To load the images, we recommend that you look into the ImageFolder dataset class of PyTorch at `https://pytorch.org/vision/main/generated/torchvision.datasets.ImageFolder.html`. The images in sub-folders of classification data are arranged in a way that is compatible with this dataset class.

4 System Evaluation

The evaluation metric is quite straightforward:

$$accuracy = \frac{\#\ correctly\ classified\ images}{\#\ total\ images} \tag{1}$$

5 Methods

If you have no idea how to start your assignment, this section can provide some suggestions.

1. The first step is to make sure you **load the datasets** correctly. You should load the training set, validation set, and test set respectively. Then train your model on the training set, and use the validation set to choose the best model. Finally, predict the labels on the test set.

[4] `https://www.kaggle.com/c/aist4010-spring2024-a1/data`

2. Start training from a simple model like **MLP** or **AlexNet**[Krizhevsky et al., 2017]. You don't have to achieve a high performance at the beginning. It's more important to **implement a pipeline**.

3. Save your results following the requirements, or you may get an extremely low score even if you have a well-trained model. I set a benchmark named **'random results'** for your reference. The predicted labels are randomly generated for the test set. So you can check if your pipeline works in the right way by comparing your results with it.

4. After you run your pipeline correctly, you can start to consider how to improve your performance. Here are some suggestions:

   - **Hyperparameter tuning.** No matter what model you choose, you can't achieve its best performance without hyperparameter tuning. You have to try different hyperparameters, including **training epochs**, **batch size**, **learning rate**, etc.
   - **Model architectures.** When you reach the limit of the model by hyperparameter tuning, introducing advanced neural networks will allow you to go further. You may find it hard to deal with image classification only by MLP, while **CNNs** are much more effective. Besides AlexNet, there are a large number of models you could try, like ResNet[He et al., 2016], DenseNet[Huang et al., 2017], SENet[Hu et al., 2018], etc. Notice that you are **NOT ALLOWED TO** directly call model APIs from libraries like torchvision model zoo[5]. But you can implement the models referring to open-source codes. Make sure you cite them in your report correctly. Besides, any pre-trained weights are **FORBIDDEN**.
   - **Data augmentation.** After trying the above two steps, you may also consider data augmentation. We know larger datasets can help improve model performance. The training set is determined since you are **NOT ALLOWED TO** find additional datasets to help train your model. But you can still use the data augmentation technique. You may crop or flip the images to generate more data. Of course, you don't need to implement these image transformations by yourself. You can refer to torchvision transforms[6].

6 Submission

The following are the deliverables for this assignment:

- **Kaggle submission.** Please submit your predicted results on the Kaggle page with your nickname. Keep it the same with A0. Make sure your nickname appears on the public leaderboard. If your score is higher than the **MLP baseline**, you can obtain at least 60%. If your score is higher than the **AlexNet baseline**, you can obtain at least 80%. The final score will depend on your ranking. At least the first three students can obtain a full score.

- **Blackboard submission.** The Deadline for the Blackboard submission is **one day later** than the Kaggle submission, so you don't need to rush. Please pack all files in one '.zip' or '.tar.gz' file named 'nickname_SID'. For example, if my nickname is 'lctest' and my SID is '1155123456', I should name my submission file as 'lctest_1155123456.zip'. And it should have these contents:

  – A report describing your model architecture, loss function, hyperparameters, and any other interesting detail led to your best result for the above competition. And cite all references in this report. Please limit the report to **two pages** (including references) and submit it in **'.pdf' format.**

  – A sub-folder containing **all** your source codes (including data-processing, training, prediction, etc.) in '.ipynb' or '.py' format.

[5] `https://pytorch.org/vision/stable/models.html`
[6] `https://pytorch.org/vision/stable/transforms.html`

7 Conclusion

Nicely done! Here is the end of Assignment 1 (Kaggle Part) and the beginning of the deep learning world. I'd like to emphasize some notices here:

- Importing additional data is **NOT ALLOWED**.
- Directly calling model APIs is **NOT ALLOWED**.
- Referring to open-source codes to implement the models is **ALLOWED**. Remember to cite them.
- Loading pre-trained weights is **NOT ALLOWED**.
- The Kaggle submission deadline is **11:59 PM (HKT), Feb. 26, 2024.** The competition will close exactly at that time. You **CANNOT** use late days for the Kaggle part assignment. Grades will be deducted by 25% for each late day.
- The Blackboard submission deadline is one day later: **11:59 PM (HKT), Feb. 27, 2024.**. You must submit the files required on Blackboard, or you will face a 10% mark deduction.

As always, feel free to ask us on Piazza if you have any questions. We are always here to help.

Good luck and enjoy the challenge!

References

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.