Homework 2 Part 2

Face Classification & Verification using Convolutional Neural Networks

11-785: Introduction to Deep Learning (Spring 2022)

OUT: **February 18, 2022**
Early Deadline/HW2P2 MCQ Deadline: **February 25, 2022, 11:59 PM**
DUE: **March 17, 2022, 11:59 PM**
Writeup Version: **1.0.0**

Start Here

- **Collaboration policy:**
  - You are expected to comply with the University Policy on Academic Integrity and Plagiarism.
  - You are allowed to talk with and work with other students on homework assignments.
  - You can share ideas but not code, you must submit your own code. All submitted code will be compared against all code submitted this semester and in previous semesters using MOSS.
  - You are allowed to help your friends debug
  - You are allowed to look at your friends code
  - You are allowed to copy math equations from any source that are not in code form
  - You are not allowed to type code for your friend
  - You are not allowed to look at your friends code while typing your solution
  - You are not allowed to copy and paste solutions off the internet
  - You are not allowed to import pre-built or pre-trained models
  - Meeting regularly with your study group to work together is highly encouraged. You may discuss ideas and help debug each other's code. You can even see from each other's solution what is effective, and what is ineffective. You can even "divide and conquer" to explore different strategies together before piecing together the most effective strategies. However, the actual code used to obtain the final submission must be entirely your own.

- **Overview:**
  - **Part 2**: This section of the homework is an open ended competition hosted on Kaggle.com, a popular service for hosting predictive modeling and data analytics competitions. The competition page can be found here and here.

  - **Part 2 Multiple Choice Questions**: You need to take a quiz before you start with HW2-Part 2. This quiz can be found on Canvas under **HW2P2: MCQ (Early deadline)**. It is **mandatory** to complete this quiz before the early deadline for HW2-Part 2.

- **Submission:**
  - **Part 2**: See the the competition page for details.

- **Note:**
  - The maximum number of parameters allowed for the models for this HW is 35 million.

Homework objective

After this homework, you would ideally have learned:

- To solve a image-based face classification problem using a CNN
  - How to set up the CNN
  - How to handle the image data
  - How to use augmentation techniques for images
  - How to train the model
  - How to optimize the model
- To explore architectures and hyperparameters for the optimal solution
  - To identify and tabulate all the various design/architecture choices, parameters and hyperparameters that affect your solution
  - To devise strategies to search through this space of options to find the best solution.
- The process of staging the exploration
  - To initially set up a simple solution that is easily implemented and optimized
  - To stage your data (e.g. by initially working on a subsample of the training data) to efficiently search through the space of solutions.
  - To track losses and performance on validation data to ensure the code is working properly and the model is being trained properly
  - To subset promising configurations/settings and then evaluate those on the larger (complete) dataset
- To engineer the solution using your tools
  - To use objects from the PyTorch framework to build a CNN.
  - To deal with issues of data loading, memory usage, arithmetic precision etc. to maximize the time efficiency of your training and inference.

1 Introduction

Even though face recognition may sound quite trivial to us humans, it remained a challenging computer vision problem in the past decades. Thanks to deep learning methods, computers now can leverage huge dataset of faces to learn rich and compact representations of human faces, allowing models to even 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**, 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 both tasks (well, other techniques are required if you want to get an A.) For the classification task, your system will be given an image of a face as input and should output the ID of the face. For the verification task, your system will be given two images as inputs and should output a score that quantifies the similarity between the *faces* in the given images. A higher score means that the faces from the two images are more likely to be from a same person.

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.

Please NOTICE: this assignment comes with **two Kaggle** competitions. In this way, you can understand how *classification* and *verification* resembles and differs from each other.

- **Face classification**
  - Goal: Given a person's face, return the ID of the face
  - Kaggle: `https://www.kaggle.com/c/11-785-s22-hw2p2-classification`
- **Face verification**
  - Goal: Given two faces, return whether they are from the same person
  - Kaggle: `https://www.kaggle.com/c/11-785-s22-hw2p2-verification`

[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] In this case, embeddings for face information.

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 assignment 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**.

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 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 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-class classification task.

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

Classic multi-class classifier = feature extractor(CNN) + classifier(FC)

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* × *num of face-ids*, followed by Softmax, to classify the image among the N (i.e., num of face-ids) 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**

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

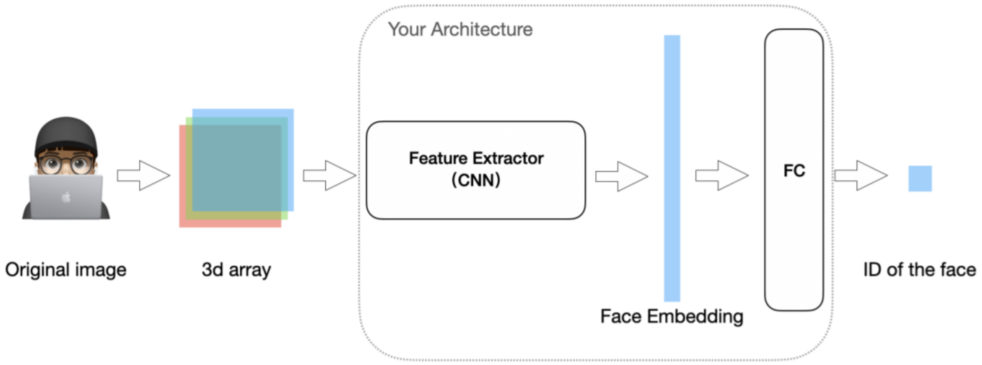

Figure 1: A typical face classification architecture

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 Face Verification

Let's switch gear to face verification. Now, 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 indicates higher confidence that the faces in the two images are of the same person.

In the following sections, we will introduce you to a few approaches. But do not let us constrain your imagination. There are a lot of other ways to achieve great performance.

FAIR WARNING: We do not guarantee that all the methods listed below can help you pass the A cut-off. You need to experiment with your own judgement.

3.1 Building upon the multi-class classification

I hope you have not deleted your classification model. If your model yields high accuracy in face classification, you **might** already have a good Feature Extractor for free. That being said, if you remove the fully connected/linear layer, this leaves you with a CNN that ”can” (*probably* can should be more accurate here) generate discriminative face embeddings given arbitrary face images.

3.1.1 Feature extractor + distance calculator

We shall all agree that the face embeddings of a same person should be similar (the distance is short) even if they are extracted from different images. Assume our CNN is competent to generate accurate face embeddings, we only need to find a proper **distance metric** to evaluate how close given face embeddings are. If two face embeddings are close [4] in distance, they are more likely to be from a same person [5].

Here, we propose two prevalent distance metrics, but you have to experiment yourself from there. (Hint: check Appendix A)

- Cosine Similarity
- Euclidean Distance

If you follow this design, your system should look like this. Please notice that the Feature Extractor in Fig 2 is the same one even though it is drawn twice.

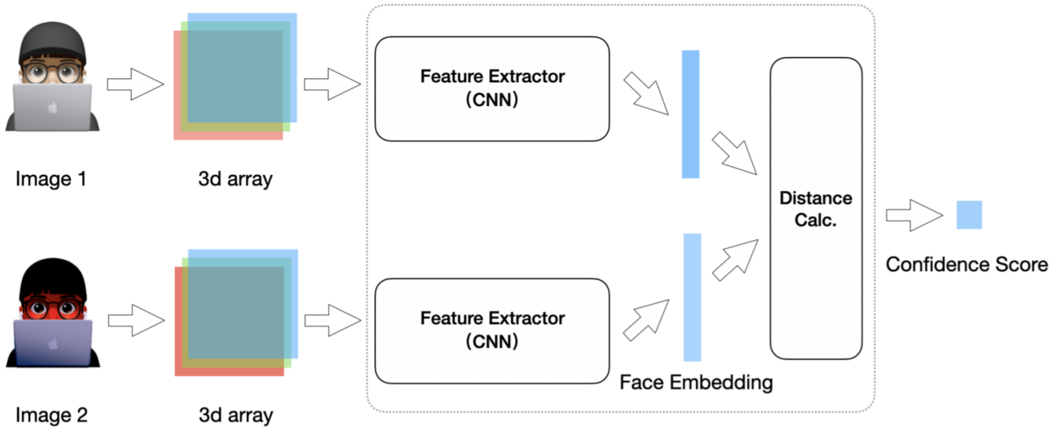

Figure 2: face verification architecture

3.1.2 Take a step further

We have heard a rumor that a good job in classification is only guaranteed to help you reach the B-cutoff in validation. Hence, you are encouraged to try other advanced loss functions such as Center-loss [1], LM [2], L-GM[3], and other architectures such as SphereFace [4], CosFace[5], ArcFace[6] and UniformFace[7] to go beyond this.

Alternatively, 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[8], pair wise loss[9].

[4] How close is close?

[5] Now, do you understand why we use fixed-length vector as face embeddings?

You are also encouraged to explore the interconnection between classification accuracy and verification performance.

3.2 Metric Learning

The multi-class classification method has a flaw here: in the real world, we can not make our model to recognize every person on Earth. What if a new person is added to the dataset? Do you want to re-train the whole network whenever a new person is added?

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 [6].

A potential approach is to build a Siamese Neural Network [10] and apply a Contrastive loss function as follows:

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

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* [7] pairs within the mini-batch, which is also referred to as **Hard Sample Mining**[11] [12].

Instead of measuring the similarity between pairs, you can also apply Triplet loss [13] or Quadruplet loss [14] to model the similarities among triplets or quadruplets.

If you're wondering if there exists a Quintuplets, Sextuplets, Septuplets or even Octuplets loss, you can refer to the N-pair Loss [15], Lifted-Structure Loss [16], Softtriplet Loss [17] papers.

It may also be possible for other advanced loss functions such as Pair-Wise Loss [18], Multi-Similarity(MS) [19], Mask Proxy(MP) [20] to give SOTA verification performance.

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

[7] Large similarity for negative pairs and small similarity for positive pairs.

4 Dataset

The data for the assignment can be downloaded from the Kaggle competition link. The dataset contains images of size 224 × 224 for all 3 channels. In this competition, we are dealing with faces from 7000 people (That being said, we have 7000 classes.)

This assignment contains 2 parts:

- 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 you need to calculate the similarity score for those images using the embeddings generated by your classification network. Notice that 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:

4.1.1 Kaggle Classification

- `classification`: Each sub-folder in `train`, `dev` 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` set to train your model **both for the classification task and verification task**.
  - `dev`: You are supposed to use `dev` to validate the classification accuracy.
  - `test`: You are supposed to assign IDs for images in `test` and submit your result.
- `classification_sample_submission.csv`: This is a sample submission file for face classification competition. The first column is the image file names. Your task is to assign ID to each image and generate submission file based on the order given here.

4.1.2 Kaggle Verification

- `dev`: This is the directory that contains the images for `Verification Validation`.
- `test`: This is the directory that contains the images for `Verification Test`.
- `verification_dev.csv`: This file contains the image pairs for `Verification Validation`. The first and second columns are the image paths of the pair. 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_test.csv`: This file contains the image pairs for `Verification Test`. The first and second columns are the image paths of the pair. You task is to compute the similarity between each two trials and to generate submission file based on this.
- `verification_sample_submission.csv`: This is a sample submission file for face verification competition. Note that the ID column should be assigned in the order of test pairs defined in `verification_test.csv`.

4.2 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/0.8/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 Multi-class classification, and Metric Learning tasks.

5 System Evaluation

5.1 Kaggle 1: Face Classification

This is quite straightforward,

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

5.2 Kaggle 2: Face Verification

Here, we 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 [8]. 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).

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

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 [9]:

- `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.

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/Autolab.

7 Conclusion

Nicely done! Here is the end of HW2P2, and the beginning of a new world. As always, feel free to ask on Piazza if you have any questions. We are always here to help.

Good luck and enjoy the challenge!

[9] `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html`

Appendix A

A.1 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:

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

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. 3. So during verification, you are strongly suggested to apply cosine similarity rather than Euclidean distance to compute the similarity score.

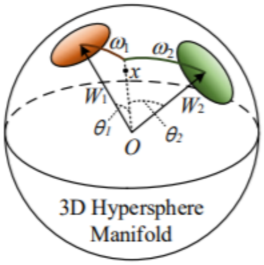

Figure 3: Angular Space [4]

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?)

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

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.

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

Appendix B

B.1 B-way Classification

In this appendix, we are going to introduce you to a **metric learning** strategy called *B-way classification*, in which B refers to batch size. The following figure gives an intuition of this manner:

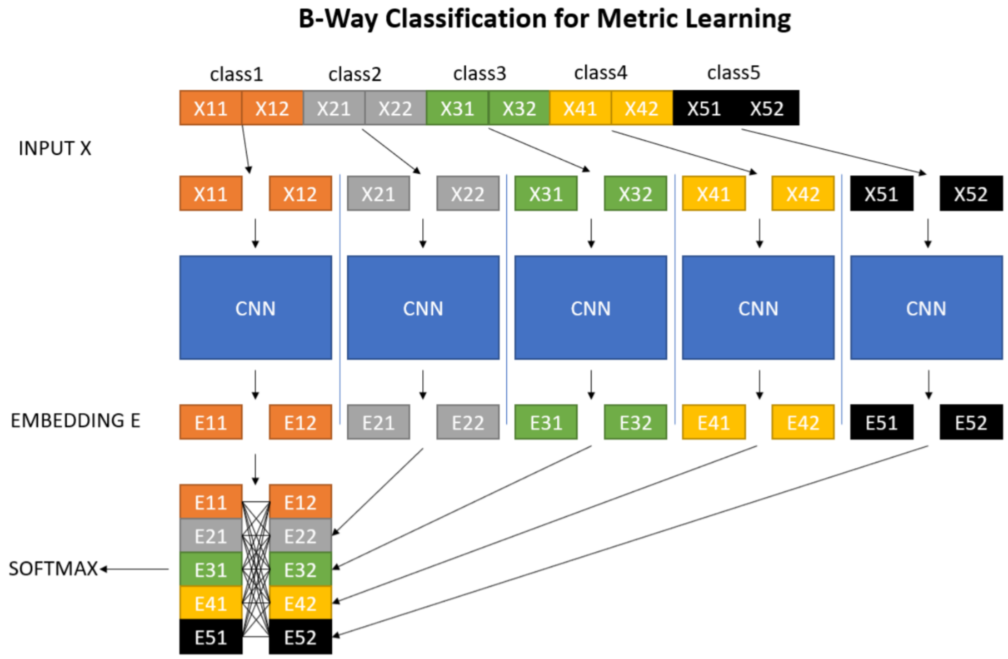

Figure 4: B-way classification for metric learning

Note that everything happens only within a mini-batch and batch size is 5 in this figure. For the notation, in X_{ij} and E_{ij} , i is the label information($1 \leq$ i $\leq$ BatchSize) and j is the index of samples for each class. Here we set number of samples in each class as 2. To claim again, if the batch size is B, then you will get 2 × B embeddings and 2 for each class. Your task is just to classify these B classes:

$$L = -\sum_{i=0}^{B-1} log \frac{e^{E_{i1}^T E_{i2}}}{\sum_{j=0}^{B-1} e^{E_{i1}^T E_{j2}}} \tag{5}$$

This is actually called *Prototypical loss*, which is one of the SOTA metric learning losses currently and which is likely to give you a better AUC score than classification methods(Even better than margin-based Softmax loss functions like CosFace/ArcFace). To apply this loss function, you may care about the following points:

- There is only one CNN backbone within a mini-batch though we present 5 in the example. (You can also apply Siamese Network)

- Label information is ignored when computing the loss objective. Labels are just 0,1,...B-1. Labels are only useful when building your dataset.
- You need to build a powerful dataset/dataloader to pass these B $\times$ 2 data points into the network, which is the most pivotal part in the whole work
- $E_{i1}^T E_{i2}$ could be replaced by $a \cdot E_{i1}^T E_{i2} + b$, in which a and b are learnable parameters. It would usually be better to normalize embeddings.
- There is no supervision signal in the loss objective unlike multi-class classification. (Is this unsupervised learning?)

Just feel free to go through this method!

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] 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] 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.

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

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

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

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

[13] 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.

[14] 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.

[15] 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.

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

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

[18] 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.

[19] 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.

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