Homework 2 Part 2

Face Classification & Verification using CNN

11-785: Introduction to Deep Learning (Fall 2024)

DUE: **12th Oct, 2024**
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 and work with other students for 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, however - 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
  - Meeting regularly with your study group to work together is highly encouraged. 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.
  - **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.

Homework objective

After this homework, you would ideally have learned:

- To implement CNNs for image data
  - How to handle image data
  - How to use augmentation techniques for images
  - How to implement your own CNN architecture
  - How to train the model
  - How to optimize the model using regularization techniques
- To derive semantically meaningful representations
  - To understand what semantic similarity means in the context of images
  - To implement CNN architectures that are one of the many ways commonly used for representation learning
  - To identify similarity or distance metrics to compare the extracted feature representations
  - To measure the semantic similarity between two derived representations using these appropriate similarity measures. Use this to generate discriminative and generalizable feature representations for data. Explore different advanced loss functions and architectures to improve the learned representations
  - Learn how classification and verification are connected
- 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
- 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

Checklist

Here is a checklist page that you can use to keep track of your progress as you go through the write-up and implement the corresponding sections in your starter notebook.

---

1. Getting started
   - ☐ Join the Kaggle competition
   - ☐ Download the starter notebook
   - ☐ Load the libraries, install Kaggle API, and download Kaggle dataset files
2. Complete the train, val, and test dataset classes, initialize the datasets and data loaders
   - ☐ Explore and visualize your dataset
   - ☐ Explore and experiment with transformations and normalization
3. Build a model
4. Run training and evaluation
   - ☐ Set up everything for training
   - ☐ Save your model checkpoints
5. Run testing and submit final predictions to Kaggle
6. (Optional) Finetune the model using different losses
   - ☐ Refer to the additional notebook for more details

Contents

1 Introduction

1.1 Overview

In this homework, we will build and train Convolutional Neural Networks (CNNs) for face recognition tasks. The most successful face recognition systems today, as of early 2019 [1, 2], can achieve more than 99% accuracy for faces of any identity. This remarkable accuracy implies that face recognition systems do not necessarily need to be trained on face images of all possible identities in the world but can still robustly conduct **open-set** inference, i.e., recognizing identities not present in the training set. Beyond face recognition, such systems can be extended to many other applications, such as retail product recognition, surveillance person identification, and vehicle identification, as long as they are trained on the specific data relevant to these tasks.

In this homework, we will learn how to build our own face recognition systems from two perspectives:

- How to build effective CNN architectures.
- How to build CNNs for face verification using different loss functions.

1.2 Architectures of CNNs

Convolutional Neural Networks (CNNs) have significantly evolved since their inception, becoming more sophisticated and efficient in handling complex tasks. Early architectures like LeNet [3] and AlexNet [4] paved the way, focusing on simple layers and large fully connected networks. Subsequent architectures, such as VGGNet [5], introduced deeper networks with uniform layer structures, while ResNet [6] introduced residual connections to enable the training of much deeper networks.

In this homework, we will focus on modern CNN architectures known for their effectiveness in face recognition tasks, such as ResNet [6] and its variants [7, 8]. These architectures are designed to extract robust features that are crucial for accurate face recognition.

1.3 Face Recognition Loss Functions

Face recognition presents unique challenges, including variations in lighting, pose, expression, and occlusion, all of which can significantly affect the performance of recognition systems. To address these challenges, the loss functions used to train face recognition models must ensure that the features extracted are not only discriminative but also invariant to these variations.

We will explore several loss functions specifically designed to enhance the performance of face recognition systems, including feature-based losses like Triplet Loss [9] and N-Pair Loss [10], as well as margin-based Softmax losses like ArcFace [2]. These loss functions aim to improve the separability of different identities in the feature space, ensuring that the system performs well even under challenging conditions.

The trained networks will be validated via face verification, where given two face images, your network needs to determine whether these two faces are from the same person identity.

2 Problem Specifics

In this assignment, you'll explore how to extract important features from face images that can be used to effectively verify identities. By the end of this homework, you'll have a better understanding of how face recognition systems work, particularly in distinguishing between different faces.For this, you will have to implement the following:

- **A face classifier that can extract feature vectors from face images.** The face classifier consists of two main parts:
  - **Feature extractor**:
    - **Objective**: Your goal is to design a model that can learn and identify distinctive facial features (like skin tone, hair color, nose size, etc.) from a person's face image. These features will be represented as a fixed-length feature vector, commonly known as a face embedding.
    - **How It Works**: To achieve this, you will explore different architectures that involve multiple convolutional layers. These layers will help your model break down the image into various components:
      - **Low-Level Features**: The first few layers might detect simple elements like edges or lines.
      - **High-Level Features**: As you stack more convolutional layers, the model will start to recognize more complex patterns by combining these low-level features, such as detecting shapes, textures, and specific facial attributes.
    - **Why It Matters**: This hierarchical decomposition of the image is essential because it allows the model to understand and capture the detailed characteristics that make each face unique.
  - **Classification Layer**:
    - **Objective**: Once you have obtained the feature vector (face embedding) from the feature extractor, you'll use it to classify the image into one of several categories (e.g., identifying the person in the image from a set of known identities).
    - **How It Works**: The feature vector will be passed through a linear layer or a Multi-Layer Perceptron (MLP), which will output the probability of the image belonging to each of the 'N' categories. The model will then use cross-entropy loss during training to optimize its performance.
    - **Why It Matters**: After training the model, these feature vectors can be used not just for classification but also for face verification tasks, where you'll compare feature vectors from different images to determine if they belong to the same person.
      Your model needs to be able to learn facial features (e.g., skin tone, hair color, nose size, etc.) from an image of a person's face and represent them as a fixed-length feature vector called *face embedding*. In order to do this, you will explore architectures consisting of multiple convolutional layers.

- **A verification system that computes the similarity between feature vectors of two images.** Essentially, the face verification system takes two images as input and outputs a similarity score that represents how similar the two images are and if they are of the same person or not. The verification consists of two steps:

  1. Extracting the feature vectors from the images.
  2. Comparing the feature vectors using a similarity metric.

  A vanilla verification system looks like this:

  1. image1 $\Longrightarrow$ feature extractor $\Longrightarrow$ feature vector1
  2. image2 $\Longrightarrow$ feature extractor $\Longrightarrow$ feature vector2
  3. feature vector1, feature vector2 $\Longrightarrow$ similarity metric $\Longrightarrow$ similarity score

  NOTE: For a better understanding of what is happening during training and inference. Please refer to fig. 1

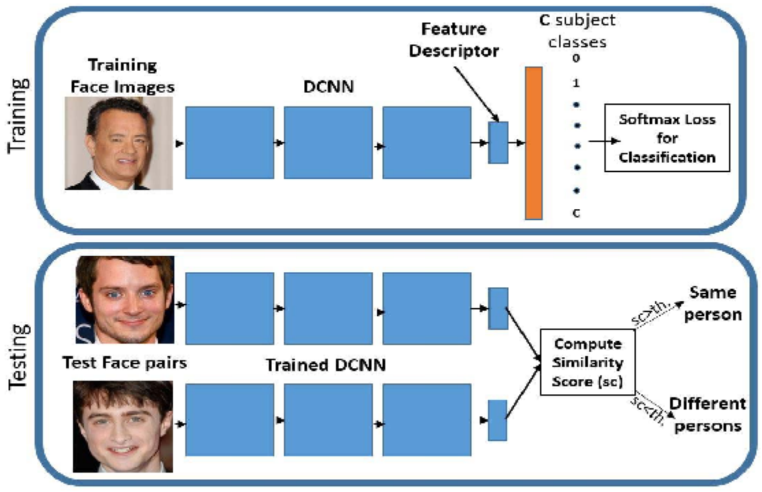

Figure 1: Face Identification and Verification setup

3 Data Description

For this homework, your training task (classification) and inference task (verification) would be are different. Thus we will have two datasets.

- The first dataset for **classification** training used in this homework is a subset of the VGGFace2 dataset. This dataset is very widely known and used in research and industry. Images are downloaded from Google Image Search and have large variations in pose, age, illumination, ethnicity, and profession (e.g., actors, athletes, politicians). The classification dataset consists of 8631 identities with image resolution of 112×112.
- The second dataset is a **verification** dataset used only for validation and testing. It consists of 6000 image pairs with 5749 identities in total.

3.1 Dataset Class - ImageFolder

Implementing the Dataset and Dataloader class for this homework is actually very straightforward: we will be using the ImageFolder class from the torchvision library and passing it the path to the training and validation dataset. The images in subfolders of `classification_data` are arranged in a way that is compatible with this dataset class. Since the folder names correspond to the classes and the images of respective classes are placed in folders with the same names, the ImageFolder class will automatically infer the labels and make a dataset object, which we can then pass on to the dataloader. The only thing to keep in mind is to include the image transforms when passing the dataset to ensure data augmentation is applied.

4 CNN Architectures and Data Augmentations

4.1 How do we differentiate faces?

Before diving into the implementation, let's pause and ask an important question: **How do we differentiate faces?**

You might think of features like skin tone, eye shapes, nose size, and other characteristics. These are known as **facial features**. These features vary significantly from person to person and are what make each of us unique. The primary goal of this assignment is to train a Convolutional Neural Network (CNN) model to identify and represent these important facial features from a person's face image. The model will do this by extracting these features and encoding them into a fixed-length vector called a **face embedding**.

Once your CNN model is capable of encoding sufficient and distinctive facial features into these face embeddings, you can then use these embeddings for further tasks, such as:

- **Face Classification**: Assigning an ID or label to a given face.
- **Face Recognition/Verification**: Identifying or verifying a person based on their face. The face could never appear in your training set.

In this assignment, you'll perform face verification by modifying your trained model. During inference, you'll remove the classification layer and use the face embeddings generated during training. These embeddings, which represent unique facial features, will be compared to determine if two face images belong to the same person, allowing you to verify identities instead of classifying them.

4.2 How do we train CNNs to produce multi-class classification?

Now comes our second question: how should we train your CNN to produce high-quality face embeddings? It may sound fancy, but conducting *face classification* is just doing a **multi-class classification**: the input to your system is a face's 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 to 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 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:

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

More concretely, your network consists of several (convolutional) layers for feature extraction. The core operation in a convolutional layer involves sliding a filter (also known as kernel) over the input data (for example an image) to produce a output feature map. The output of the last such feature extraction layers (i.e. the final output feature map) would be the face embedding. You will pass this face embedding through a linear layer whose dimension is *embedding dim* $\times$ *num of face-ids* to classify the image among the N (i.e., num

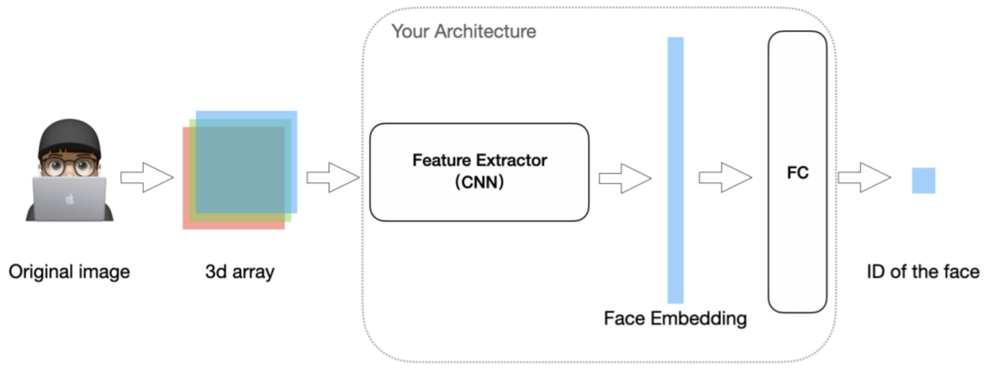

Figure 2: A typical face classification architecture

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 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**.

4.3 Transformations and Data Augmentation

When working with images in deep learning, it's essential to prepare and augment your data to improve your model's performance. PyTorch provides a module called **torchvision.transforms** that is specifically designed for this purpose. This module includes a variety of pre-defined transformations that can be easily applied to images or entire datasets. Some of the most commonly used transformations include **resizing, cropping, flipping, rotating, adjusting brightness/contrast, normalizing pixel values, and converting images to tensors**.

4.3.1 Why Transformations Matter:

Image transformations, often referred to as data augmentation techniques, play a crucial role in training Convolutional Neural Networks (CNNs). While they don't directly generate new features, they offer several significant benefits:

- **1. Increasing Data Volume**: By applying transformations to your images, you effectively increase the size of your dataset. This helps the model learn from a wider variety of examples, which can improve both training and generalization.
- **2. Preventing Overfitting**: Overfitting occurs when a model memorizes the training data instead of learning to generalize. By exposing the model to slightly altered versions

of the same images, transformations help the model focus on learning robust features rather than specific details of individual images.

- **3. Invariance**: Transformations teach your model to recognize objects regardless of their orientation, position, or other variations. This means that your model becomes more versatile and capable of handling real-world scenarios where such variations are common.
- **4. Better Generalization**: A diverse training set, created through transformations, exposes your model to a wider range of scenarios. This can lead to better performance when the model encounters new, unseen data.

4.3.2 Important Considerations:

While transformations can be highly beneficial, they should be chosen carefully to suit your specific task. Here are some key points to keep in mind:

- **Appropriate Transformations**: Ensure that the transformations you use are relevant to the type of data you're working with. For instance, if you're working with face images, applying vertical flips might not be appropriate, as upside-down faces are uncommon in real-world data.
- **Impact on Training Time**: Some transformations, especially those that require significant processing, can increase the training time per epoch. It's important to balance the benefits of augmentation with the computational cost.
- **Normalization**: Many models require the input data to be normalized. This involves subtracting the **mean** from each pixel and dividing by the **standard deviation**. This process ensures that the data is on a similar scale, which can help with model convergence. For more on how to normalize images in PyTorch, you can refer to this guide [1]. PyTorch's **torchvision.transforms.Normalize()** can be used for this purpose, but make sure to check whether it takes in images or tensors as input.

However, it's important to note that while transformations can be very beneficial, they should be chosen carefully. There are potential downsides to using image transformations. These include increased training time due to more data, the risk of inappropriate transformations depending on the context, possible distortion or loss of information, the need for careful parameter choice, and not entirely resolving overfitting issues, especially in cases of very small datasets or complex models.

4.4 Create deeper layers with residual networks

Having a network that is good at feature extraction and being able to efficiently train that network is the core of the classification task. This homework requires training deep neural

[1] How to Normalize Images in PyTorch

networks and, as it turns out, deep neural networks are difficult to train, because they suffer from vanishing and exploding gradients types of problems. Here we will learn about **skip connections** that allow us to take the activations of one layer and suddenly feed it to another layer, even much deeper in the network. Using that, we can build **residual networks** (like ResNets), which enable us to train very deep neural networks, sometimes even networks of over one hundred layers.

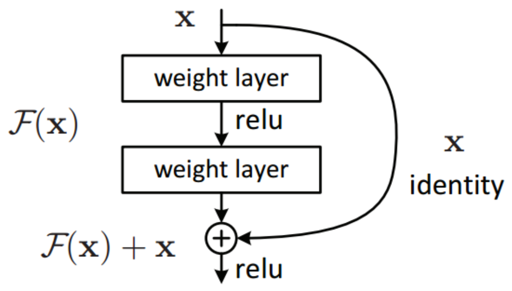

Figure 3: A Residual Block

Resnets are made of something called residual blocks, which are a set of layers that are connected to each other, and the input of the first layer is added to the output of the last layer in the block. This is called a **residual connection**. This identity mapping does not have any parameters and is just there to add the input to the output of the block. This allows deeper networks to be built and trained efficiently.

Several other blocks make use of residual blocks and residual connections and can be used for the classification task, such as **ResNet, SEResNet, ConvNext, MobilNet, etc**. *You are encouraged to read their respective research papers to understand better how they work and implement blocks from these architectures.*

4.4.1 ResNet

ResNet models were proposed in "Deep Residual Learning for Image Recognition". Here we have the 5 versions of ResNet models, which contain 18, 34, 50, 101, and 152 layers, respectively. Detailed model architectures can be found in the paper linked above.

4.4.2 ConvNeXt

ConvNeXt is a recent CNN architecture that uses inverted bottlenecks inspired by the Swin Transformer, residual blocks, and depthwise separable convolutions instead of regular convolutions. A comparison of the ResNet-50 and ConvNeXt-T and the detailed architecture can be found in "A ConvNet for the 2020s".

Since you will be using blocks and customized versions of these architectures, the performance may or may not match the expected outcomes based on the benchmarks in the papers. Thus, you are encouraged to explore various architectures systematically to get you to the high cut-off. That's pretty much everything you need to know for your Classification Kaggle competition. Go for it!

5 Face Recognition

To develop a robust face recognition system, it is essential to understand how the training data and model architecture are defined and interconnected.

Let us define the training dataset as $\mathcal{D}_{\text{train}} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i\in[N]}$, where $\mathbf{x}_i \in \mathbb{R}^{H\times W\times C}$ indicates an input face image, $\mathbf{y}_i$ is the corresponding one-hot ground-truth identity for the image, i.e., a one-hot class label, and N denotes the dataset size. We can define our face recognition system as the stack of a CNN-based feature extractor $\mathbf{z} = f_\theta(\mathbf{x})$ with learnable parameters θ, which transforms the input images $\mathbf{x}$ into features $\mathbf{z} \in \mathbb{R}^D$, where D indicates the dimension of the feature space, and a linear projection head $\mathbf{s} = g_\omega(\mathbf{z})$ with learnable parameters ω, which maps the features $\mathbf{z}$ into scores $\mathbf{s} \in \mathbb{R}^C$ with dimension C.

To express the relationship between the feature vectors and the scores in matrix form, we can write:

$$\begin{aligned} \mathbf{z}_i &= f_\theta(\mathbf{x}_i) \\ \mathbf{s}_i &= g_\omega(\mathbf{z}_i) = \mathbf{z}_i W \end{aligned} \tag{1}$$

Here, $W \in \mathbb{R}^{D\times C}$ is the weight matrix of the linear projection head, which maps the feature vector $\mathbf{z}_i \in \mathbb{R}^D$ to the score vector $\mathbf{s}_i \in \mathbb{R}^C$. Each element s_i^j in the score vector represents the unnormalized logit for class j, which will be used to compute the probability of the input image belonging to each identity class through the Softmax function.

5.1 Training Face Classifiers with Softmax Cross-Entropy

In this part, we will show the first modeling of a face recognition system as a face identity classifier. As you learned from the class, this is as easy as solving a classification problem using a feature extractor of CNN and a classification head with learnable weights $W \in \mathbb{R}^{D\times C}$[2], by optimizing the Cross-Entropy (CE) loss with the Softmax function. Recall that, in the classification setting, we usually term the score vectors $\mathbf{s}$ as **logits**, and the Softmax function converts them into probabilities:

$$\mathbf{p} = \text{Softmax}(\mathbf{s}) = [\frac{\exp(s^1)}{\sum_{k=1}^C \exp(s^k)}, ..., \frac{\exp(s^C)}{\sum_{k=1}^C \exp(s^k)}] \in \mathbb{R}^C, \tag{2}$$

where each element in the vector indicates the (predicted) probability of this sample belong to a class, e.g., $p^j = \frac{\exp(s^1)}{\sum_{k=1}^C \exp(s^k)}$ is the (predicted) probability of a sample belonging to class k. We optimize the Cross-Entropy (CE) to encourage the probability of each sample $\mathbf{x}_i$ belonging y_i to be maximized, i.e, the cross-entropy between the predicted probability and the one-hot ground truth to be minimized:

$$\mathcal{L}_{\text{CE}} = \frac{1}{N} \sum_i^N \sum_j^C -y_i^j \log p_i^j. \tag{3}$$

[2]Here we omit the bias term in the linear layer, but it can be easily incorporated into W.

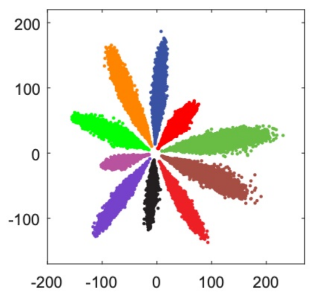

Figure 4: Feature visualization of Softmax CE loss.

With proper CNN architecture and data augmentation, we can obtain a strong classification baseline for this homework.

However, there are two main limitations of treating face recognition as a classification problem. The first limitation is obvious. The trained face classifier can only recognize the identities of the faces y that are present in the training data. During inference, we predict the identity of the face as $\arg\max_j p^j$ for the testing input. This means the system only works for closed-set identities, and when we want the system to recognize new open-set identity that is not included in the training set, we need to re-train the whole system. This becomes extremely un-affordable for million-scale and billion-scale face recognition systems, the most common scenarios in reality. One can circumvent this limitation by performing face recognition from the trained feature extractor only, without using the classification head. This requires us to compare the features of the testing input with all the features of the training data, and return the identity/label of the training sample with the largest similarity of the features: $\arg\max_{y_i} \text{Similarity}(\mathbf{z}, \mathbf{z}_i)$. The second limitation is that the feature extractor trained by classification with CE loss and Softmax presents a very poor similarity measure, which is less obvious and we will elaborate in section 5.2.

5.2 A Closer Look: What Are We Learning with CE?

$$\begin{aligned}
l_{\text{CE}} &= \sum_{j}^{C} -y^j \log p^j \\
&= \sum_{j}^{C} -y^j \log \frac{\exp(s^j)}{\sum_{k=1}^{C} \exp(s^k)} \\
&= -\log \frac{\sum_{k=1,k\neq y}^{C} \exp(s^k)}{\exp(s^y)} \\
&= s^y - \log \sum_{k=1,k\neq y}^{C} \exp(s^k) \\
&\approx s^y - \max_{k\in[C],k\neq y} (s^k)
\end{aligned} \tag{4}$$

The objective of Softmax CE loss: optimizing the ground-truth score s^y to be larger than the maximum of non-ground-truth score $\max_{k\in[C],k\neq y}(s^k)$.

We have $s^j = \mathbf{z}\mathbf{w}_j^T = \|\mathbf{z}\|\|\mathbf{w}_j^T\|\cos\theta_j$. This indicates that Softmax CE loss will strengthen the length of feature vector and the weight vector, resulting in a radial feature space, as shown in fig. 4. A radial feature space presents very limited discriminative features for feature matching of face recognition. For example, $\mathbf{z}_1$ and $\mathbf{z}_2$ belong to the same class, but it present small feature similarity compared to $\mathbf{z}_1$ and $\mathbf{z}_3$ due to the strengthed features.

5.3 Objectives of Face Recognition

The primary goals for an effective open-set face recognition system are as follows:

- **Maximize Intra-Class Similarity:** Ensure that face images of the same identity have highly similar feature representations. This means that features extracted from different images of the same person should be close together in the feature space, which facilitates accurate identification.

- **Minimize Inter-Class Similarity:** Ensure that face images of different identities have distinct feature representations. This involves making the features of different individuals sufficiently different, thereby reducing the chance of mis-identification.

- **Maintain a Large Margin between Intra-Class and Inter-Class Similarity:** Ensure that the similarity between features of the same class (intra-class similarity) is significantly higher than the similarity between features of different classes (inter-class similarity). This is crucial for robust face recognition, especially when dealing with challenging cases such as similar-looking images taken under varying conditions.

To mathematically capture these objectives, we define a loss function that penalizes small intra-class distances and large inter-class distances. A common formulation is:

$$\mathcal{L} = \max(s^n - s^p + m, 0), \tag{5}$$

where:

- $s^p = \text{Similarity}(\mathbf{z}_i, \mathbf{z}_j^p)$ represents the similarity score between the anchor feature $\mathbf{z}_i$ and a positive feature $\mathbf{z}_j^p$ from the same identity.

- $s^n = \text{Similarity}(\mathbf{z}_i, \mathbf{z}_k^n)$ represents the similarity score between the anchor feature $\mathbf{z}_i$ and a negative feature $\mathbf{z}_k^n$ from a different identity.

- m is a margin parameter that enforces a gap between the positive and negative similarity scores.

The objective of this loss function is to ensure that s^n (the similarity with a negative sample) is at least m units less than s^p (the similarity with a positive sample). If this condition is not met, the loss will be positive and the model will be penalized, thereby pushing the features to satisfy this margin constraint. By optimizing this loss, the model learns to create a feature space where same-identity faces are clustered together, and different-identity faces are well separated. Also, from this objective, we can observe that in purely CE training, the loss functions only encourage the positive similarity s^y to be larger than the maximal of the negative similarities $s^k, k \in [C], k \neq y$, without explicitly encourage a margin.

In the next, we will revisit the two paradigms of achieving this objective: feautre-based loss functions (also known as metric learning) and margin-based loss functions. We will also go through a unified view of both paradigms at the end.

5.4 Feature-based Loss Functions

5.4.1 Overview

Feature-based loss functions, also known as metric learning losses, focus on directly optimizing the feature space to enhance the discriminative power of the learned representations. Instead of merely classifying faces into predefined categories, these loss functions aim to structure the feature space such that similar identities are clustered closely together while dissimilar identities are pushed far apart. This approach is particularly advantageous for open-set face recognition, where the system encounters identities not seen during training.

5.4.2 Triplet Loss

Triplet loss [9] is one of the most widely used feature-based loss functions. It operates on triplets of images: an anchor image $\mathbf{x}_i$, a positive image $\mathbf{x}_i^p$ (from the same identity as the anchor), and a negative image $\mathbf{x}_i^n$ (from a different identity). The goal of the triplet loss is to ensure that the distance between the anchor and the positive image is smaller than the distance between the anchor and the negative image by at least a margin α. The triplet loss function is defined as:

$$\mathcal{L}_{\text{triplet}} = \sum_{i=1}^{N} \left[\|f(x_i^a) - f(x_i^p)\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + m \right]_+, \tag{6}$$

where $f(x)$ denotes the feature representation of image x learned by the model, $\|\cdot\|_2^2$ is the squared Euclidean distance (Cosine similarity can also be used, in which case the sign of the similarity score would be flipped), and $[\cdot]_+$ denotes the hinge loss that only penalizes the loss when it is positive.

This loss encourages the model to learn a feature space where:

- The features of images from the same identity (anchor and positive) are close together.
- The features of images from different identities (anchor and negative) are far apart by at least the margin m.

Looking closer at the Triplet loss function, you will find it is very similar to what was defined earlier as the objective for face recognition, i.e., eq. (5), by replacing the similarity metric as the Euclidean distance and flipping the sign.

5.4.3 N-Pair Loss

N-Pair loss [10] generalizes the concept of triplet loss by considering multiple negative samples for each anchor-positive pair, rather than just one. This allows for more robust training by incorporating a broader context of the feature space. The N-Pair loss is defined as:

$$\mathcal{L}_{\text{N-Pair}} = \sum_{i=1}^{N} \log \left[1 + \sum_{j \neq i} \exp \left(f(x_i^a)^T f(x_j^n) - f(x_i^a)^T f(x_i^p) \right) \right], \tag{7}$$

where $f(x_i^a)^T f(x_j^n)$ denotes the similarity between the anchor and a negative sample, and $f(x_i^a)^T f(x_i^p)$ denotes the similarity between the anchor and the positive sample.

The N-Pair loss effectively encourages the model to maximize the similarity between the anchor and the positive pair while minimizing the similarity with **multiple negatives** (compared to only one negative in triplet loss), providing a more comprehensive training signal.

5.4.4 Remark

Feature-based loss functions are powerful for structuring the feature space, but they come with certain challenges:

- **Directly Optimizing the Objective:** You will find these metric loss direcly optimize the objectives we listed earlier.
- **Difficult to Tune:** Choosing the right margin parameter α in triplet loss or selecting the appropriate number of negative samples in N-Pair loss can be challenging and may require extensive hyper-parameter tuning. Usually, **it makes life easier to combine the metric learning loss with CE loss**.
- **Hard Sample Mining:** The effectiveness of feature-based losses heavily relies on the selection of hard samples—triplets or pairs where the model struggles to differentiate between positive and negative samples. Efficient mining strategies are crucial but can be computationally expensive.
- **Training Complexity:** These loss functions often require careful batching and sampling strategies to ensure the training process is stable and effective, adding complexity to the model training pipeline.

Despite these challenges, feature-based loss functions are indispensable for applications requiring robust face recognition, especially in open-set scenarios where the system must generalize to unseen identities.

5.5 Margin-based Softmax Loss Functions

5.5.1 Overview

Margin-based Softmax loss functions are designed to enhance the discriminative power of features learned by deep neural networks, particularly in classification tasks like face recognition.

These loss functions modify the traditional Softmax Cross-Entropy (CE) loss by introducing a margin that separates different classes more distinctly in the feature space. This approach is particularly useful in scenarios where fine-grained distinctions between classes (e.g., different identities) are critical, as it encourages the model to maximize inter-class variance while minimizing intra-class variance.

5.5.2 ArcFace

ArcFace is one of the most popular margin-based Softmax loss functions. It introduces an angular margin to the Softmax function, which helps in learning more discriminative features by pushing the decision boundaries further away from each other. The formulation of ArcFace is as follows:

Given the feature vector $\mathbf{z}_i$ of a sample and the corresponding weight vector $\mathbf{w}_j$ for class j, the original Softmax function is defined as:

$$s^j = \mathbf{z}_i \cdot \mathbf{w}_j = \|\mathbf{z}_i\| \|\mathbf{w}_j\| \cos\theta_j, \tag{8}$$

where θ_j is the angle between the feature vector $\mathbf{z}_i$ and the weight vector $\mathbf{w}_j$.

ArcFace modifies this by adding an **angular margin** m to θ_j, effectively making it harder for the model to classify samples correctly, thus pushing the model to learn more discriminative features. To make the angular margin effective, the magnitude of both feature and weight vectors are normalized to unit length and scaled by a fixed parameter α , which transforms the logit (similarity score) to:

$$s^j = \gamma\cos(\theta_j + m) = \alpha\left(\cos\theta_j \cos m - \sin\theta_j \sin m\right). \tag{9}$$

The modified logits are then passed through the Softmax function as usual:

$$\mathbf{p} = \text{Softmax}(s^j), \tag{10}$$

The key idea is that by introducing the margin m, the model is forced to learn features that not only separate classes, but also create a margin between them in the angular space, leading to more robust face recognition.

5.5.3 Combined Margin-based Loss Functions

In addition to ArcFace, other margin-based loss functions such as CosFace [1], SphereFace [11], and AM-Softmax [12] have been proposed, each introducing different types of margin (additive angular, multiplicative angular, etc.). These can be combined to create a unified margin-based loss function that takes advantage of multiple margin types simultaneously.

For instance, a combined margin-based loss function could be formulated as:

$$s^j = \gamma(\cos(\theta_j + m_1) + m_2), \tag{11}$$

where m_1 and m_2 are different types of margins (e.g., angular and additive). This allows for more flexibility in training and can lead to improved performance in specific face recognition tasks. You are encouraged to try different combination of margin and find out which one works the best.

5.5.4 Remark

Margin-based Softmax loss functions have several advantages:

- **Easier to Train:** Compared to feature-based loss functions, margin-based Softmax losses are easier to implement and train because they do not require complex sample mining strategies. They directly modify the logits before applying the Softmax function, which simplifies the training pipeline.
- **Effective for Fine-Grained Classification:** By introducing a margin, these loss functions make the decision boundaries between classes more distinct, which is crucial for tasks like face recognition where classes (identities) can be very similar.
- **Limited to One Positive Class:** A limitation of these methods is that they are designed for single-label classification scenarios, where each sample belongs to only one class. They are less effective in multi-label scenarios or when the task requires recognizing multiple faces simultaneously.
- **Scalability:** Margin-based Softmax loss functions scale well with the number of classes, making them suitable for large-scale face recognition systems.

In summary, margin-based Softmax loss functions strike a balance between ease of training and the ability to learn highly discriminative features, making them a popular choice for modern face recognition systems.

5.6 A Unified View of Feature-based and Margin-based Softmax Loss Functions

5.6.1 Positives vs. Negatives

Both feature-based and margin-based Softmax loss functions share a common goal: to enhance the discriminative power of the learned features by maximizing the similarity between samples of the same class (positives) and minimizing the similarity between samples of different classes (negatives). While they approach this objective differently, the underlying principles are closely related.

Feature-based loss functions, such as Triplet Loss and N-Pair Loss, explicitly operate on the distances or similarities between anchor-positive and anchor-negative pairs. They directly optimize the feature space to ensure that positive pairs are close and negative pairs are far apart.

Margin-based Softmax loss functions, like ArcFace, introduce a margin in the classification layer to implicitly achieve a similar effect. By modifying the logits before applying the Softmax function, these losses create a separation (or margin) between different classes, pushing the model to learn more distinct features. The weight of the last linear classifier can be viewed as an anchor for each classes (especially after being normalized), which thus ties the margin-based Softmax loss functions closely with feature based loss functions. In the next, we will introduce more modern loss functions that provides a unified view of both paradigms.

5.6.2 Circle Loss

Circle Loss [13] is an example of a unified approach that combines elements of both feature-based and margin-based losses. It introduces a unified perspective by directly optimizing the similarity scores with a margin, while also ensuring that the optimization is focused on hard samples (those that are difficult to classify). Circle Loss is defined as:

$$\mathcal{L}_{\text{circle}} = \log \left[1 + \sum_{\text{pos}} \exp(\gamma(\cos\theta_{\text{pos}} - m)) \sum_{\text{neg}} \exp(\gamma(\cos\theta_{\text{neg}} + m)) \right], \tag{12}$$

where γ is a scaling factor, m is the margin, θ_{pos} and θ_{neg} are the angles corresponding to positive and negative pairs, respectively.

Circle Loss unifies the optimization of intra-class and inter-class similarities by applying different margins to positive and negative pairs, ensuring that the learned feature space is well-structured for discriminative tasks like face recognition.

5.6.3 Loss Function Rewrite

Both feature-based and margin-based losses can be rewritten to highlight their commonalities. For instance, the triplet loss can be seen as a special case of a margin-based loss when we view the distance metric as a form of similarity score with a margin. Similarly, margin-based Softmax losses can be interpreted as imposing a margin directly on the logit-level comparisons that occur in feature-based losses.

This unified perspective allows us to see that both approaches are variations of the same fundamental principle: enforcing a margin between positive and negative samples to improve feature discriminability.

5.6.4 Supervised Contrastive Loss

Supervised Contrastive Loss (SupCon) [14] is another loss function that unifies the principles of feature-based and margin-based approaches. It leverages contrastive learning, a self-supervised technique, in a supervised setting to enhance intra-class compactness and inter-class separability. The SupCon loss is defined as:

$$\mathcal{L}_{\text{supcon}} = \frac{1}{N} \sum_{i=1}^{N} \frac{-1}{|P(i)|} \sum_{p \in P(i)} \log \frac{\exp(\mathbf{z}_i \cdot \mathbf{z}_p / \tau)}{\sum_{a=1}^{2N} \mathbb{I}_{[a \neq i]} \exp(\mathbf{z}_i \cdot \mathbf{z}_a / \tau)}, \tag{13}$$

where $\mathbf{z}_i$ and $\mathbf{z}_p$ are the feature representations of the anchor and positive samples, τ is a temperature scaling parameter, and $P(i)$ denotes the set of positives for anchor i.

Supervised Contrastive Loss encourages samples from the same class to be pulled together in the feature space, while samples from different classes are pushed apart. This aligns with the goals of both feature-based and margin-based losses, providing a robust approach to learning discriminative features.

5.7 Remark

Although these loss functions provide a unified view of both learning paradigms, they usually require a very large batch size (>2048) to achieve reasonable performance. We suggest that you try these loss functions with multiple GPU cards or in combination with previous loss functions. You will be able to achieve a high enough score with previous loss functions only.

6 Face Verification

Now let us switch gears to face verification. After training of your model using the loss functions, the input to your system will now be a pair of face images that may or may not belong to the same person during inferece. Given a *pair*, your goal is to output a numeric score that quantifies how similar the faces in the two images are. A higher score indicates a higher confidence about whether the faces in the two images are of the same person.

6.1 Building upon the multi-class classification

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 could* would be more accurate here) generate discriminative face embeddings, given arbitrary face images.

6.2 Verification Pipeline

We shall all agree that The face embeddings of the same person should be similar (i.e., the distance between feature vectors generated is small), even if they are extracted from different images. After being trained explicitly to maximize the similarity of positive pairs, minimize the similarity of negative pairs, and encourage some margin between positives and negatives, your model is capable to generate accurate face embeddings. We only need to compute a proper **distance metric** to evaluate how close the given face embeddings are. If two face embeddings are close in distance, they are more likely to be from the same person.

If you follow this design, your system should look like the Figure below. Please notice that the Feature Extractor in Figure 5 is the same one, even though it is drawn twice.

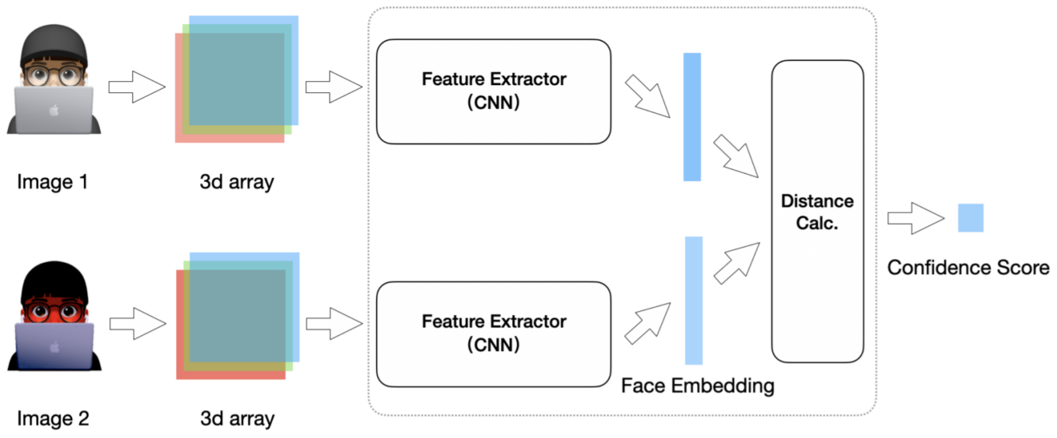

Figure 5: Face verification architecture

7 Kaggle Competitions

For this assignment, we will provide **one kaggle** competition for the face verification task. But we will provide validation data for both classification and verification tasks. One for testing your classifier on *closed-set* identities, i.e., same as your training set, and another for testing your verification system on *open-set* identities, i.e., different from your training set. The purpose of the first classification competition is for you to test your classifier and get a sense that higher classification accuracy may not necessarily imply a good verification system (as shown in the CE loss part).

- **Face classification**
  - Goal: Given a person's face, return the identity of the face.
- **Face verification**
  - Goal: Given a list of known and unknown identities, map each unknown identity to either a known identity or a special, "no-correspondence" label.
  - Kaggle: `https://www.kaggle.com/competitions/11785-hw-2-p-2-face-verification-fa`

7.1 File Structures

The structure of the dataset folders is as follows:

7.1.1 Classification Dataset Folder

- Each sub-folder in `train` and `dev` 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.

7.1.2 Kaggle Verification dataset folder

- `ver_data`: This is the folder of all face images with unknown identities.
- `test_pair.csv`: This is the test file where each two sampled images are treated as a pair. And you need to predict a score for each pair for the kaggle to compute the metric.
- `verification_sample_submission.csv`: This is a sample submission file for the face verification competition. The first column is the index of the image files. Your task is to assign a label to each image and generate a submission file as shown here.

7.1.3 Evaluation System

- **Face Classification**
  This is quite straightforward,

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

- **Kaggle: Face Verification**
  In this task, the performance is evaluated using the Equal Error Rate (EER). The EER is the point at which the rate of false acceptances (False Acceptance Rate, FAR) equals the rate of false rejections (False Rejection Rate, FRR). These rates are defined as follows:

$$\text{FAR} = \frac{\text{\# of false acceptances}}{\text{total \# of impostor attempts}}$$

$$\text{FRR} = \frac{\text{\# of false rejections}}{\text{total \# of genuine attempts}}$$

  The EER is the value where these two rates are equal, indicating the threshold at which the system's error rates are balanced. A lower EER indicates better performance.

$$\text{EER} = \text{FAR} = \text{FRR}$$

8 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!

Appendix A

A.1 List of relevant recitations

Please review the below recitations for supplementary material that could be helpful for this assignment -

- Pytorch Fundamentals
- OOPS Fundamentals
- Google Colab
- GCP
- Kaggle
- Data Loaders
- WandB
- Blocks coding
- Discriminative Losses

References

[1] Hao Wang, Yitong Wang, Zheng Zhou, Xing Ji, Dihong Gong, Jingchao Zhou, Zhifeng Li, and Wei Liu. Cosface: Large margin cosine loss for deep face recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5265–5274, 2018.

[2] Jiankang Deng, Jia Guo, Niannan Xue, and Stefanos Zafeiriou. Arcface: Additive angular margin loss for deep face recognition. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4690–4699, 2019.

[3] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[5] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[6] 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.

[7] 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.

[8] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11976–11986, 2022.

[9] Elad Hoffer and Nir Ailon. Deep metric learning using triplet network. In *Similarity-based pattern recognition: third international workshop, SIMBAD 2015, Copenhagen, Denmark, October 12-14, 2015. Proceedings 3*, pages 84–92. Springer, 2015.

[10] Kihyuk Sohn. Improved deep metric learning with multi-class n-pair loss objective. *Advances in neural information processing systems*, 29, 2016.

[11] 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.

[12] Feng Wang, Jian Cheng, Weiyang Liu, and Haijun Liu. Additive margin softmax for face verification. *IEEE Signal Processing Letters*, 25(7):926–930, 2018.

[13] Yifan Sun, Changmao Cheng, Yuhan Zhang, Chi Zhang, Liang Zheng, Zhongdao Wang, and Yichen Wei. Circle loss: A unified perspective of pair similarity optimization. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 6398–6407, 2020.

[14] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning. *Advances in neural information processing systems*, 33:18661–18673, 2020.