Homework 2 Part 2

Face Classification & Verification using CNN

11-785: Introduction to Deep Learning (Fall 2023)

OUT: **25th September, 2023**
Early Deadline/HW2P2 MCQ Deadline: **5th October, 2023**
DUE: **22nd October, 2023**
Writeup Version: **1.0.1**

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 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.

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 a 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 each other
- 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. As you complete each function in the notebook, you can check the corresponding boxes aligned with each section. It is recommended that you go through this write-up and starter notebook simultaneously, step by step.

1. Getting Started
    - ☐ Download starter notebook and set up the virtual environment (optional)
    - ☐ Install Kaggle API and create a directory
    - ☐ Download dataset files from Kaggle
2. Complete the training loop *train_model()*
    - ☐ Create the dataloader for the training dataset
    - ☐ Set model in 'Training Mode'.
    - ☐ Clear the gradients
    - ☐ Compute the model output and the loss
    - ☐ Complete the Backward Pass
    - ☐ Update model weights
3. Complete inference loop *evaluate_model()*
    - ☐ Set model to 'Evaluation Mode' and create validation set dataloader
    - ☐ Compute model output in "no grad" mode
    - ☐ Calculate validation loss using model output
    - ☐ Get most likely class as a prediction from the model output
    - ☐ Calculate the classification validation accuracy
    - ☐ Calculate similarity using a similarity metric and get the most likely identity
    - ☐ Calculate the verification validation accuracy
4. Complete Early Submission Quiz
    - ☐ Complete HW2P2 Early Submission Quiz
5. Classification Hyper-parameter Tuning
    - ☐ Make initial submission before the early submission deadline
    - ☐ Use Weights and Biases to log metrics for each epoch
    - ☐ Make sure the model is saved after every few epochs
    - ☐ Iterate with different hyper-parameters to reach desired cut-offs

Contents

1 Introduction

Implementation: Your solution should implement CNN-based architectures using CNN blocks from architectures like (but not limited to) ResNet, MobileNet, EfficientNet or ConvNext. You can search online for other blocks that might work better for this homework and implement them. You can also combine different blocks together and test them. The parameter limit for this homework is 21M. Refer to Appendix A for details on what Recitations to look at before beginning the implementation.

Restrictions: You may not use any data besides that provided as a part of this homework. You are not allowed to use pretrained models, or use models without implementing the code yourself (e.g importing models from torchhub, or copying code from public repositories). You're supposed to limit the number of parameters in your model to 21 million. You will not get marks for this homework if you exceed this limit.

1.1 Executive Summary

In this homework you will work on pattern recognition problems that require **position invariance**. Specifically you will work on the problem of recognizing or verifying faces in images. In typical pictures of faces, the face is rarely perfectly centered. Different pictures of the same person may have the face shifted by varying amounts. The classifier must recognize the face regardless of this indeterminacy of position. This calls for position-invariant models, specifically Convolutional Neural Networks, or CNNs.
A CNN is a neural network that derives representations (or embeddings) of input images that are expected to be invariant to the precise positions of the patterns in it. These embeddings are subsequently classified by downstream classifiers (which may just be an additional softmax layer, or even an MLP, appended after the convolutional layers), to achieve position-invariant pattern recognition.

1.2 Overview

In this homework, you will learn to build CNN-based architectures for face classification and face verification. The homework will instruct you on two key concepts:

- How to build effective convolutional neural networks.
- How to generate discriminative and generalizable feature representations for data.

1.2.1 Face Classification

Face classification is a *closed set* multiclass classification problem, where the subjects in the test set have also been seen in the training set, although the precise pictures in the test set will not be in the training set. For this to achieve high accuracy it is only required that the embeddings for (all pictures of) the subjects in our "vocabulary" be linearly separable from each other.

1.2.2 Face Verification

Face verification refers to the task of determining whether two face images are of the same person, without necessarily knowing who the person is. Face verification is an instance of a larger class of problems where we attempt to determine if two data instances belong to the same class without necessarily knowing (or having a model for) the class itself.

This is a common problem used in a variety of situations, for instance when your laptop uses facial recognition to identify you. You would have "enrolled" yourself with an enrollment image, and later when you try to login, your system compares a picture it takes of your face to the stored enrollment image to determine if both are from the same person. This is also the same approach that the FBI uses to identify suspects from facial images, when it compares the pictures to a catalogue of pictures of known subjects, to determine if the captured image is a match to any of them.

1.2.3 Differences between classification and verification

In a verification you are given an exemplar of a category of data (e.g. a face), and an unknown instance, and you must determine if the two are from the same class. Why is this not a *classification* problem, where the exemplar is the training instance to train a classifier from, and the unknown instance is the test instance? The obvious response – that a single training instance is insufficient to train a classifier – is not the entire answer. The answer lies in the definition of the *negative* instances – training data that are *not* from the class, that are also required to train a classifier. One may try to randomly draw data from other classes as negatives, but now we are faced with three problems:

- Since we do not know *what* the class is, we cannot be sure (without additional information or labelling) if the negative samples we have drawn are indeed negative, i.e. they don't belong to the same class as our exemplar;
- The space of negatives (all possible images) is so large that any sample of negatives may not cover it sufficiently and result in a biased classifier that will incorrectly accept some types of negatives since they were not part of our sample set;
- Finally, the expense of training an entire classifier for each "match" problem may not be justified, in many situations. E.g., if you were using matching to perform retrieval from a database, you would need to train a classifier for every instance in your database, which seems excessive. E.g., for a database of a billion instances, you would need a billion classifiers.

So we turn the problem around – instead of asking, "is this test instance closer to the target class than it is to the negative class" (which is what a classifier does, effectively), we ask, "is the test instance close enough to the target class to declare a match", without reference to the negative class. We need a robust and accurate way of determining if an unknown instance is close enough to an exemplar.

The idea is to train a model to extract discriminative feature vectors from images, which have the property that feature vectors from any two images that belong to the same person

are closer than feature vectors derived from images of two different persons. Once we have trained such a model, the solution is simple – given any pair of facial images, we will extract feature vectors from both and compute their similarity (according to our metric). If the similarity exceeds a threshold, we will declare a match.

2 Problem specifics

In this homework, you will learn how to extract discriminative features from face images that can be used to achieve a good performance in face verification. 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**: 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.

    Stacking several convolutional layers allows for hierarchical decomposition of the input image. For example, if the first layer extracts low-level features such as lines, then the second layer (that acts on the output of the first layer) may extract combinations of low-level features, such as features that comprise multiple lines to express shapes.
  - **Classification layer**: The feature vector you obtain at the end will then be passed through a linear layer or a MLP to classify it among 'N' categories, and use cross-entropy loss for optimization. The feature vectors obtained after training such a model can then be used for the verification task.
- **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 $\implies$ feature extractor $\implies$ feature vector1
  2. image2 $\implies$ feature extractor $\implies$ feature vector2
  3. feature vector1, feature vector2 $\implies$ similarity metric $\implies$ similarity score

  Important: We have framed the problem a bit differently as shown in Fig 1, as a one-to-many comparisons, where we compare one image to many images and then predict the image with the highest similarity; instead of just comparing two images at a time and predicting if they are of the same person or not, we are going to compare each unknown identity with all the known identities and then decide whether this unknown identity matches any of the known identity using a threshold method, and predict the known identity with the highest similarity score.

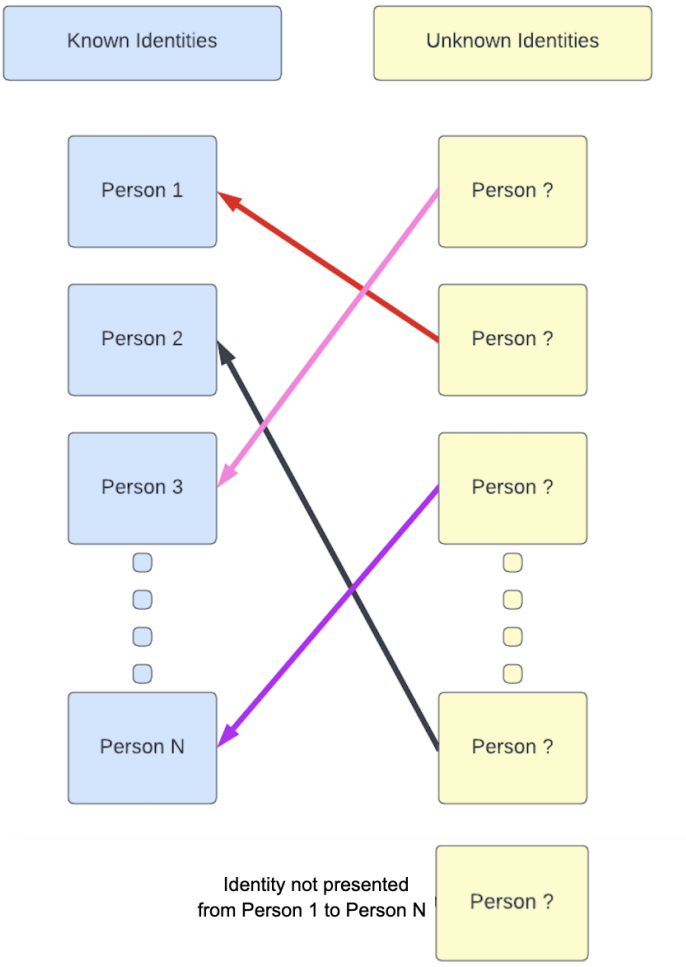


Figure 1: Modified Verification task: Mapping of the unknown identities to known identities

3 Data Description

The dataset being 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 dataset was collected with three goals in mind:

1. To have a large number of both identities and images per identity.
2. To cover a large range of pose, age, and ethnicity.
3. To minimize the label noise.

The classification dataset consists of 7,001 identities. The verification dataset consists of 1080 identities. The dataset has been class-balanced, so each class has the equal number of training images, and all the images are resized to 224 x 224 pixels.

To summarize, this assignment contains 2 parts:

- For **classification**, you will be given an image of a human face. What you need to do is to learn to classify this image with the correct face identity from 7001 identities.
- For **verification**, you will have 1080 unknown identity images, split into 360 for the Dev-set and 720 for the Test-set. Each of these unknown identity images will need to be mapped either to one of 960 known identities, or to n000000, the "no correspondence" label, for the remaining 120 identities.
  - **Dev-set**: Each of the 360 unknown identity images in the Dev-set will either have an identity label or a "no correspondence" label (n000000).
  - **Test-set**: The 720 unknown identity images in the Test-set will not have any corresponding, true identity label.

  Note: The dataset is not a one-to-one mapping – that is, the unknown images don't map to a unique image in the known folder.

3.1 Dataset Class - ImageFolder

Implementing the Dataset and Dataloader class for this homework is actually very straight forward: 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 dataloder. The only thing to remember is to also pass the image transforms to the dataset class for doing data augmentation.

4 Kaggle Competitions

For this assignment, you will compete in **two Kaggle** competitions. In this way, you can understand how *classification* and *verification* tasks resemble and differ from each other.

- **Face classification**
  - Goal: Given an person's face, return the identity of the face.
  - Kaggle: `https://www.kaggle.com/competitions/11-785-f23-hw2p2-classification`
- **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/11-785-f23-hw2p2-verification`

4.1 File Structures

The structure of the dataset folders is as follows:

4.1.1 Kaggle Classification dataset folder

- 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. Note that you should assign IDs in the range of [0, 7000]. `ImageFolder` dataset by default maps each class to such an ID and you can rely on that.
- `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 a label to each image and generate a submission file as shown here.

4.1.2 Kaggle Verification dataset folder

- `known`: This is the directory of all 960 known identities.
- `unknown_test`: This is the directory that contains the images for `Verification Test`. There are 720 images of unknown identities here, which are demographically balanced.
- `unknown_dev`: This is the directory that contains 360 images of unknown identities which you are given the ground truth mapping for.

- `verification_dev.csv`: This is a list of ground truth identity labels (each label maps to a known identity in the known folder or the "no correspondence" label) for the sorted list of images in the unknown_dev folder. This will help you calculate the dev accuracy for verification.

- `verification_sample_submission.csv`: This is a sample submission file for 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.

4.1.3 Evaluation System

- **Kaggle 1: Face Classification**
  This is quite straightforward,

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

- **Kaggle 2: Face Verification**
  This is also quite straightforward,

$$\text{accuracy} = \frac{\#\text{ correctly matched unknown identities}}{\text{total unknown identities}}$$

5 Face Classification

5.1 How do we differentiate faces?

Before we dive into the implementation, let us 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's face image. These extracted features will be represented in a *fixed-length* vector of features, known as **face embeddings**.

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.

5.2 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:

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

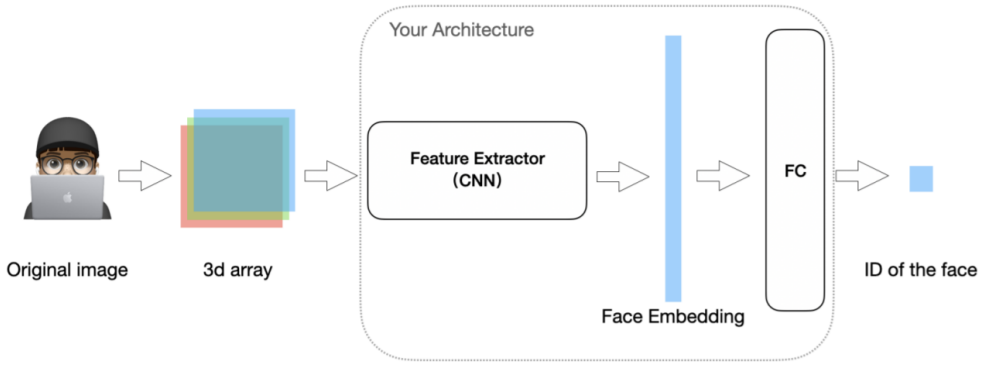


Figure 2: A typical face classification architecture

More concretely, your network consists of several (convolutional) layers for feature extraction. The input will be (possibly a part [1] of) the image of the face. The output of the last such feature extraction layers 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 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**.

5.3 Transformations and Data Augmentation

PyTorch offers a module called `torchvision.transforms` specifically designed for image transformations. This module provides a wide range of pre-defined transformations that can be easily applied to images or datasets. Some common transformations include **resizing, cropping, flipping, rotating, adjusting brightness/contrast, normalizing pixel values, and converting images to tensors**.

Image transformations such as these, also known as data augmentation techniques, play a crucial role in training Convolutional Neural Networks (CNNs) and improving their performance. They don't directly contribute to feature generation, but they provide several key advantages:

1. More Data: Transformations increase the dataset size by applying various alterations to input images, leading to improved training and generalization.

2. Preventing Overfitting: Transformations expose the model to 'new' images each epoch, reducing memorization of specific images and promoting the learning of robust features.

3. Invariance: Training the model on transformed images teaches it to recognize objects irrespective of their orientation or position (invariance).

4. Better Generalization: Transformations diversify the training set, exposing the model to a wide variety of examples, which can enhance performance on unseen data.

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 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.

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

Also, transformations should reflect the types of variation you expect in your real-world data. For example, if you're working with images of faces, using vertical flips as a transformation might not make much sense, since upside-down faces are relatively rare in real-world scenarios. You may take a look at the PyTorch transformations [2] and experiment further to see how they affect the results.

Note: Some models require the data to be normalized before being input. If you do not normalize the image data you could run into convergence issues even if your model is implemented perfectly. Data normalization is done by subtracting the **mean** from each pixel and then dividing the result by the **standard deviation**. Each channel (RGB) would have a separate mean and standard deviation (you may refer to [3] to see how you can do this). Once you have figured out how to calculate these values, you can use `torchvision.transforms.Normalize()`. Read the PyTorch documentation to see if it takes in images or tensors.

5.4 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 to train deep neural 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 (resnets)**, which enable us to train very deep neural networks, sometimes even networks of over one hundred layers.

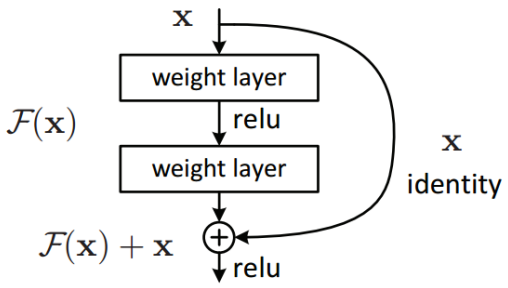


Figure 3: A Residual Block

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.

[2] https://pytorch.org/vision/stable/transforms.html
[3] https://www.geeksforgeeks.org/how-to-normalize-images-in-pytorch/

There are several other blocks that make use of residual blocks and residual connections and can be used for the classification task, such as **MobilNet, ResNet and ConvNext etc**. *You are encouraged to read their respective research papers to understand better how they work and implement blocks from these architectures.* You can then combine different blocks and test what combination works the best. Two papers are mentioned below.

5.4.1 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.

5.4.2 ConvNeXt

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

Since you will only be using blocks and not the entire architecture, these may or may not be able to get you to the high cut-off, so you are encouraged to explore other architectures as well as combination of different blocks. That's pretty much everything you need to know for your Classification Kaggle competition. Go for it!

6 Face Verification

Now let us switch gears to face verification. The input to your system will now 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 a higher confidence about whether the faces in the two images are of the same person.

6.1 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.

6.2 An architectural design suggestion

We shall all agree that the face embeddings of the same person should be similar (the distance between feature vectors generated is small) even if they are extracted from different images. Assuming our CNN is able 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 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 4 is the same one, even though it is drawn twice.

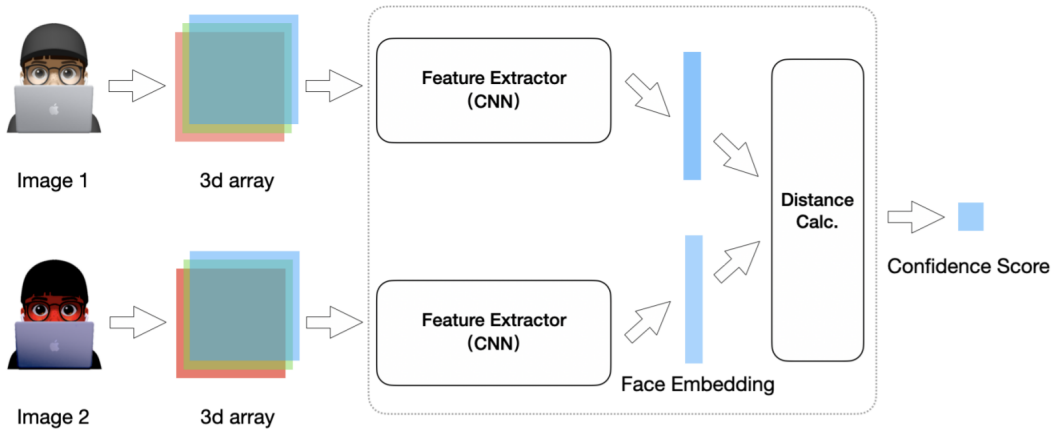


Figure 4: face verification architecture

6.3 Similarity metric

As mentioned before, we need a distance metric to compare how "close" two feature vectors are in order to verify if they belong to the same subject. The comparison is done by using a similarity metric, which is a function that takes two feature vectors as input, and outputs a number that represents how similar the two feature vectors are: If the number is high, then the two feature vectors are similar, and if the number is low, then the two feature vectors are not similar.

The way this problem is set up is that you will be given a dataset of images, half of which are known identities and the other half of unknown identities. Some of the unknown identities have a one-to-one mapping with the known identities, and some of them do not. Identities that do not have a mapping in known identities will have a low similarity value with all known identities – therefore, you first need to set a threshold: If your maximum similarity score between an unknown identity and every known identity is below this threshold, we can say that the unknown identity is not presented in the known set. Otherwise, for those similarity scores that are above the threshold, your job is to predict the correct mapping. Essentially, for each unknown identity, you will have to predict the known identity that it corresponds with (i.e. with highest similarity value) or, if the similarity score is below the threshold, predict that it is not represented in the known set.

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

- Cosine Similarity
- Euclidean Distance

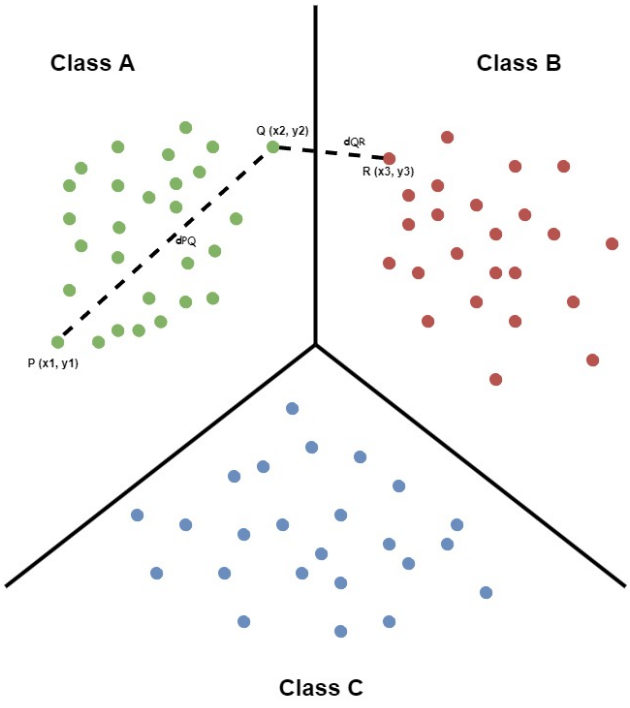
6.4 Advance Loss and more

We have heard a rumor that a good job in classification is only guaranteed to help you reach the medium cutoff in verification. Hence, you are encouraged to try other advanced loss functions such as Center-loss [1], triplet-loss[8], pair wise loss[9], LM [2], L-GM[3], and other architectures such as SphereFace [4], CosFace[5], ArcFace[6] and UniformFace[7] to go beyond this.

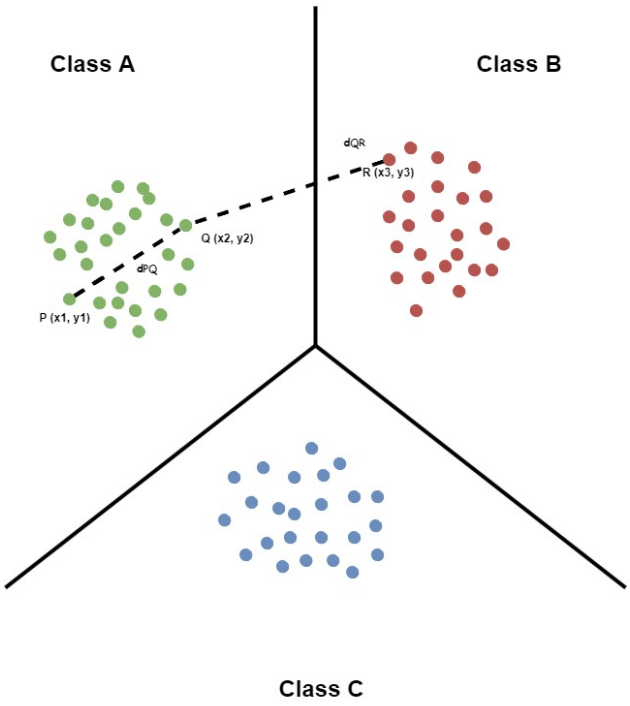
6.4.1 Contrastive losses

Although face classification is a good alternate task for face verification, crossing the high cutoff for classification does not guarantee that you will also cross the high cutoff in face verification. Let us see why this might happen using a toy example:

Say your dataset has face images belonging to 3 classes: A, B, and C. Each point in Figure 1 represents a feature vector produced for an image after training. From figure 1, we can see that using approach 1; you will be able to produce feature vectors that are 'separable' by optimizing the network using cross-entropy loss.



(a) Not optimized for direct comparison of feature vectors.



(b) Optimized for direct comparison of feature vectors.

However, this may not be enough for the face verification task in this homework. This is because this approach does not optimize for direct comparison of two instances (feature vectors) to see if they belong to the same class. It may be possible that the distance between feature vectors belonging to the same class (i.e. dPQ in Fig 5a) is greater than the distance between feature vectors belonging to different classes (i.e. dQR) even though the classes are separable. This is because minimizing cross-entropy loss only aims to make the classes linearly separable in the embedding space. It does not guarantee production of highly discriminative feature vectors.

In order to resolve this issue and enhance the discriminative power of feature vectors, their intra-class compactness and inter-class variability need to be simultaneously maximized. In other words, the distance dPQ (between feature vectors belonging to the same class) as shown in Fig 5a needs to be minimized (for increasing compactness within class A) and the distance dQR (between feature vectors belonging to different classes) needs to be maximized (to increase inter-class variability between classes A and B). The important thing to note here is that P and Q are the farthest points within class A. Our goal is to develop a model such that even the distance between the farthest points in each class is less than the distance between points belonging to 2 different classes.

Several advanced loss functions have been proposed to encourage discriminative learning of features like Center loss, Sphere loss, Large-margin softmax loss, Large-margin Gaussian mixture loss etc. Each of these losses is jointly used with cross-entropy loss to get high-quality feature vectors. After training, the feature vectors will look as shown in Figure 5b. Comparing the two figures, you can see that the feature vectors in the same class are closer and the ones in different classes are farther.

6.4.2 Pairwise Loss

After implementing the contrastive loss, you may wonder: What if the resulting embeddings do not generalize well to new faces in the test set? You are right. In simple face verification problems, testing classes may be present in the training set (also called closed set identification), and contrastive loss alone will be enough for such simple problems. But in more complex, real-world face verification problems, you will encounter faces in the test set that belong to classes your model has never seen before. The feature vectors learned during training need to be not only separable but also discriminative and generalized enough to identify unseen classes in the test set.

A better approach, in this case, would be to train a model to directly optimize the face embeddings without explicit reference to their classes. The resulting network may be even more efficient.

This optimization can be achieved by using 'pair-wise' loss functions, which involve computing similarities between embeddings of pairs of inputs, with the objective of maximizing the similarity of instances belonging to the same class while minimizing that of instances that belong to different classes. *You are encouraged to look into losses like Triplet loss for this approach.*

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!

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

A.2 Cosine Similarity VS Euclidean Distance

You may struggle with selecting a proper distance metric for the verification task. The two most popular distance metrics used in verification are cosine similarity and Euclidean distance. Both metrics are able to reach state-of-the-art score for this homework, but you should get an intuition on when, for what kind of problem, to choose one or the other. The metric should be training-objective-specific, where training objective refers to the loss function.

Let us start with revisiting Softmax cross entropy, where Y_i is the label of X_i:

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

If you take a thorough look at this formula, you will find that the objective is to make the vector(embedding) X_i closer to the vector W_{Y_i} and further 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, each embedding would be similar to its center in the **Angular Space**, as illustrated in Figure 7. So during verification, you are strongly suggested to apply cosine similarity rather than Euclidean distance to compute the similarity score.

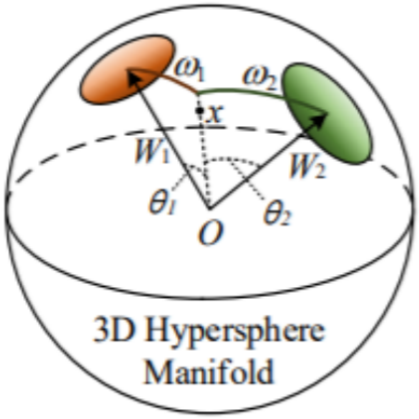


Figure 6: Angular Space [4]

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 a radial basis function?)

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

A question we leave to you: what metric is probably better if you were to start with metric learning and apply the loss function shown in Eq. 2?

However, the aforementioned conclusions are not definitively true. Sometimes Euclidean distance is also good when you apply softmax XE in Eq. 3 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.

$$||U - V||_2^2 = ||U||_2^2 + ||V||_2^2 - 2U^T V \tag{3}$$

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`.