Homework 2 Part 2

Face Classification & Verification using Convolutional Neural Networks

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

OUT: **September 30, 2022**
Early Deadline/HW2P2 MCQ Deadline: **October 10, 2022, 11:59 PM**
DUE: **October 26, 2022, 11:59 PM**
Writeup Version: **1.1.0**

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

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

1 Introduction

Key new concepts: Face classification and verification, convolutional neural networks, metric learning.

Implementation: Your solution should implement CNN-based architectures including but not limited to ResNet, MobileNet or ConvNeXt. You can search online for other architectures that might work better for this homework and implement them.

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).

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.

We will consider two kinds of problems in this setting.

In the first, *classification*, we will attempt to identify the person in a picture. This is a *closed set* 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.

In the second, *verification*, we will attempt to only determine if the person in a given "query" picture is also present in a given gallery of images or not, with no reference to their identity. This is a *open set* problem, where the subjects in the test data may not have been seen during training at all. In order to solve this problem we will require that the embeddings of two pictures of the same person are always closer than those from pictures of two different people. With such embeddings, in order to solve the gallery problem we must only determine if the embedding of any of the pictures in the gallery is sufficiently close to that of the query.

So, in this homework, we will learn to deal with two new concepts: position-invariant pattern classification, and how to build classifiers/detectors for novel classes that are not present in our training data.

1.2 Overview

In this homework, you will learn to build CNN-based architectures for 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. A note on the problem

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.

Such problems arise in many situations.

1. Information retrieval (e.g., Google search by image): This is a service provided by Google that allows a user to search for images using an image as the starting point, rather than a written or spoken search query. When you input the query image, Google will try to find images similar to it or exact copies based on the recognition of the subject within it. For example, if you provide an image of an animal or plant (that you do not know anything about) as a query, Google will return several similar images showing the same subject. This is based on the same principle of image matching (based on its features) that we use in face verification.

2. Speaker verification: We know that a speaker's voice contains personal traits of the speaker, given the unique pronunciation organs and speaking manner of the speaker, e.g., the unique vocal tract shape, accent, rhythm, etc. Therefore, it is possible to automatically identify a speaker from his/her voice using deep learning. In speaker verification, the input is a query spoken utterance whose identity is confirmed by comparing it to a gallery of known speakers and getting the best match. It has a wide range of applications, such as the voice-based authentication of personal smart devices, such as cellular phones, vehicles, and laptops. It is also important in audio-based information retrieval for broadcast news, meeting recordings, and telephone calls.

In each case, you will note that the problem proceeds as follows: you are given an exemplar of a category of data (e.g., a face), and a "probe" instance, and you must determine if the two are from the same class. Why is this not a *classi cation* problem, where the exemplar is the training instance to train a classifier from, and the probe is the test instance (and the "matching" task is of classifying the probe instance with the classifier trained from the exemplar)? 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, of course, try to randomly draw data from other classes as negatives, but now we are faced with two 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.

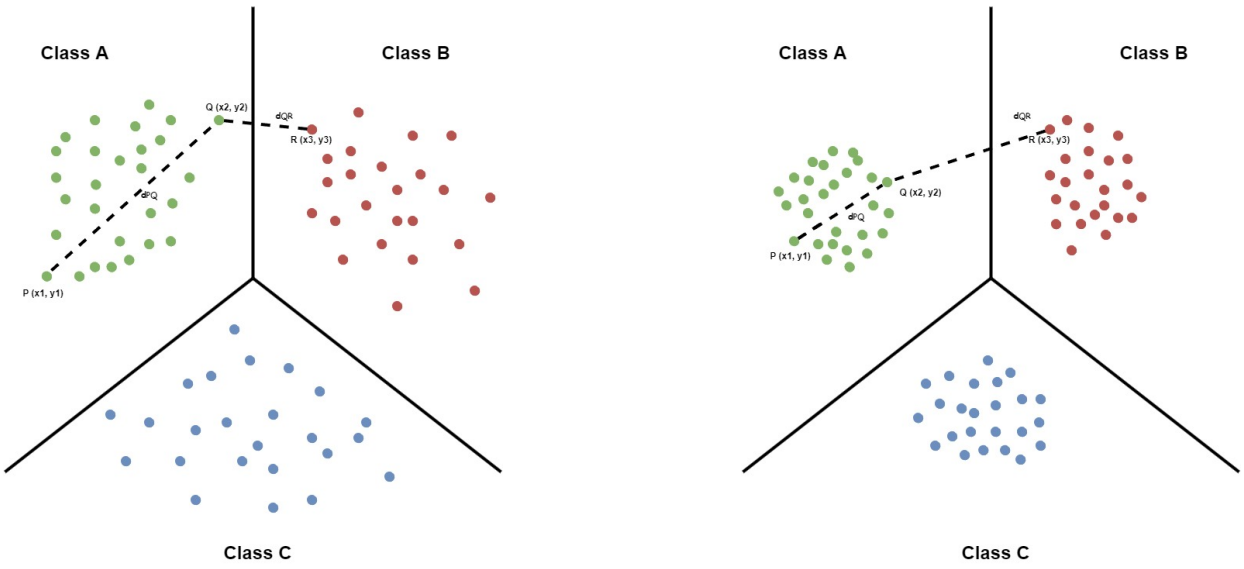
So we turn the problem around instead: instead of asking, "is this test instance closer to the target class than it is to the negative class" (which is what a classifier does, effectively), we ask, "is the test instance close enough to the target class to declare a match", without reference to the negative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

The idea is to train a model to extract discriminative features (feature vectors) from images, which have the property that feature vectors from any two images that belong to the same person are closer (according to some metric) 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. Now, let us discuss a few technical approaches to solve the problem of face verification:

1. **Approach 1:** As mentioned earlier, the key aspect of the face verification task is to be able to generate discriminative feature vectors from a person's face. Since you are measuring the closeness (using some metric) between these features obtained from two faces, you also can put them into categories/classes based on that closeness. In other words, face images of persons with similar features (i.e features for which closeness is below a certain cutoff) will belong to the same class, and face images of persons with different features (i.e features for which closeness is above a certain cutoff) will belong to different classes. So what if we treat the verification task as a classification one in order to produce high-quality feature vectors (or face embeddings)? Since classifying an image into one of several classes also requires the model to extract discriminative features from it, our problem will become much more simpler.

   In this approach, we will perform multi-class face classification where the input to your system will be an image of a person's face, and the model will predict the class(out of a total of N classes) that image belongs. In the ideal case, a face classification model should be able to classify any person's face correctly, but that is an extremely difficult task requiring huge amounts of data. In this homework, you will work with a very large dataset called "VGGFace2" to produce "good" feature representations.

   Your network will consist of several convolutional layers, the input will be a person's image from "VGGFace2", and the output will be a feature vector or face embedding. You will then pass this feature vector through a linear layer followed by Softmax to classify it among N categories. Basically,



Multi-class classifier = Feature extractor (CNN layers) + Classifier (FC layer)

You can then use cross-entropy loss to optimize your network to predict the correct person for every training image.

2. **Approach 2:** Although face classification is a good alternate task (as explained in approach 1) 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. 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 1) is greater than the distance between feature vectors belonging to different classes (i.e. dQR in Fig 1) 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 1 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 2. Comparing figures 1 and 2, you can see that the feature vectors in the same class are closer and the ones in different classes are farther.

3. **Approach 3:** After going through the first two approaches, 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 approach 2 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 than the ones used in approaches 1 and 2.

   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, contrastive loss, and quadruplet loss for this approach.

1.3 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 - the feature extractor and the classification layer.

  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.

  You will then pass this feature vector (obtained at the end of through a linear layer followed by Softmax 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.** The face 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 $=)$ feature extractor $=)$ feature vector1
  2. image2 $=)$ feature extractor $=)$ feature vector2
  3. feature vector1, feature vector2 $=)$ similarity metric $=)$ similarity score

  So essentially, the 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. We have framed the problem a bit differently; 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 predict the known identity with the highest similarity score. So we have framed the problem as a one-to-many comparisons, where we compare one image to many images and then predict the image with the highest similarity. Even though this is a bit different from the original problem, it is still a verification problem, and the same verification system can be used to solve both problems. Most of the moving parts are the same.

- **Training.** And last, but not the least you will learn how to train complicated models that can perform verification as well as classification.

This may seem like a lot, but believe it or not, you will manage to get all this done in the course of this homework. In the following sections, we will outline how.

We provide you a baseline architecture based on ResNet, from "Deep Residual Learning for Image Recognition" and describe the key components of it in Section 4.3. However, you are also welcome to try other architectures like ConvNeXt, MobileNet, and EfficientNet.

2 Kaggle

For this assignment, you will compete in **two Kaggle** competitions. In this way, you can understand how *classi cation* and *veri cation* 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-f22-hw2p2-classification`

- **Face verification**
  - Goal: Given a list of known and unknown identities, map each unknown identity to a known identity.
  - Kaggle: `https://www.kaggle.com/competitions/11-785-f22-hw2p2-verification`

2.1 File Structure

The structure of the dataset folder is as follows:

2.1.1 Kaggle Classification

- `classification`: Each sub-folder in `train`, `dev` and `test` contains images of one person, and the name of that sub-folder represents their ID.
  - `train`: You are supposed to use the `train` set to train your model **both for the classification task and verification task**.
  - `dev`: You are supposed to use `dev` to validate the classification accuracy.
  - `test`: You are supposed to assign IDs for images in `test` and submit your result. Note that you should assign IDs in the range of [0, 6999]. `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.

2.1.2 Kaggle Verification

- `known`: This is the directory of all known identities that we know.
- `unknown_test`: This is the directory that contains the images for `Verification Test`. There are 800 images of unknown identities here, which are demographically balanced.
- `unknown_dev`: This is the directory that contains 200 images of unknown identities which you are given the ground truth mapping for.
- `dev_identities.csv`: This is a list of ground truth identity labels (each label maps to a known identity in the known folder) for the sorted list of images in the unknown_dev folder. This will help you calculate the dev accuracy for verification.

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 7,000 identities. The verification dataset consists of 1000 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 7000 identities.
- For **verification**, you will have 1000 unknown identity images (split into dev and test) and they need to be mapped one of the 1000 known identities. For the dev-set, you will be given 200 unknown identities and you will be expected to find known identities (out of 1000 known identities) that these unknown identities map to. For these dev images you will also be given the identity label of the image that it maps to in the known images. Similarly, for the test set you will have 800 unknown identities that would need to be mappped to their corresponding known identities, except you will not know what identity it maps to. Note: It 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 (`https://pytorch.org/vision/0.8/datasets.html#imagefolder`) and passing it the path to the training and validation dataset, because the folder names corresponds 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, details on how to implement that can be found in Pytorch documentation. The images in subfolders of `classification_data` are arranged in a way that is compatible with this dataset class. Note that `ImageFolder` is helpful for both Multi-class classification, and Metric Learning tasks.

4 Face Classification

4.1 Face Embedding

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 *xed-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.

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

4.2 Multi-class Classification

It may sound fancy, but conducting *face classi cation* 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:

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

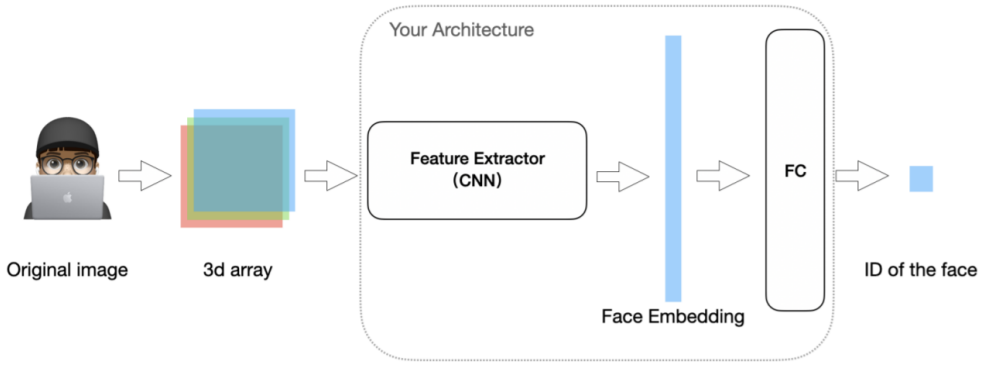


Figure 1: A typical face classification architecture

More concretely, your network consists of several (convolutional) layers for feature extraction. The input will be (possibly a part [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

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

embedding through a linear layer whose dimension is *embedding dim* $\times$ *num of face-ids*, followed by a Softmax, to classify the image among the N (i.e., num of face-ids) people. You can then use cross-entropy loss to optimize your network to predict the correct person for every training image.

The ground truth will be provided in the training data (making it supervised learning). You are also given a validation set for fine-tuning your model. Please refer to the **Dataset 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 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 very 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 a hundred layers.

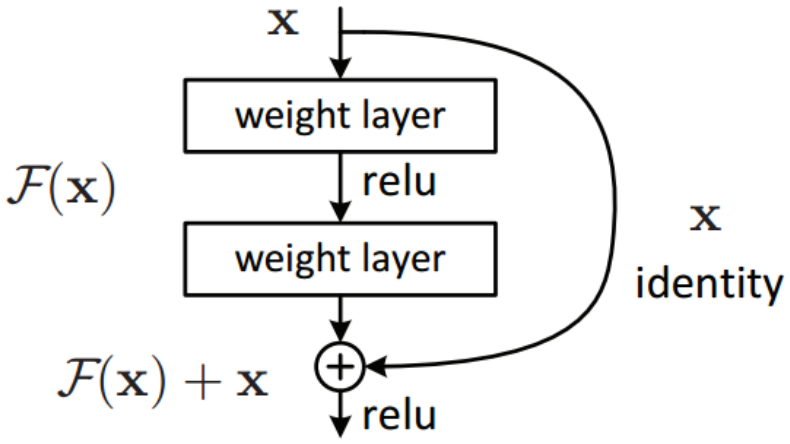


Figure 2: A Residual Block

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.

There are several popular architectures 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. Two of the most popular ones are mentioned below, for your reference.

4.3.1 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.

4.3.2 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".

These may or may not be able to get you to the high cut-off, and there are many other architectures that may give you better results, so you are encouraged to explore other architectures as well. That's pretty much everything you need to know for your Classification Kaggle competition. Go for it!

5 Face Verification

Let us switch gear to face verification. Now, the input to your system will be a *trial*, i.e., a pair of face images that may or may not belong to the same person. Given a *trial*, your goal is to output a numeric score that quantifies how similar the faces in the two images are. A higher score indicates higher confidence that faces in the two images are of the same person.

5.1 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.

5.1.1 Feature extractor + distance calculator

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 competent to generate accurate face embeddings; we only need to find a proper **distance metric** to evaluate how close given face embeddings are. If

two face embeddings are close [2] in distance, they are more likely to be from the same person [3].

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

- Cosine Similarity
- Euclidean Distance

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

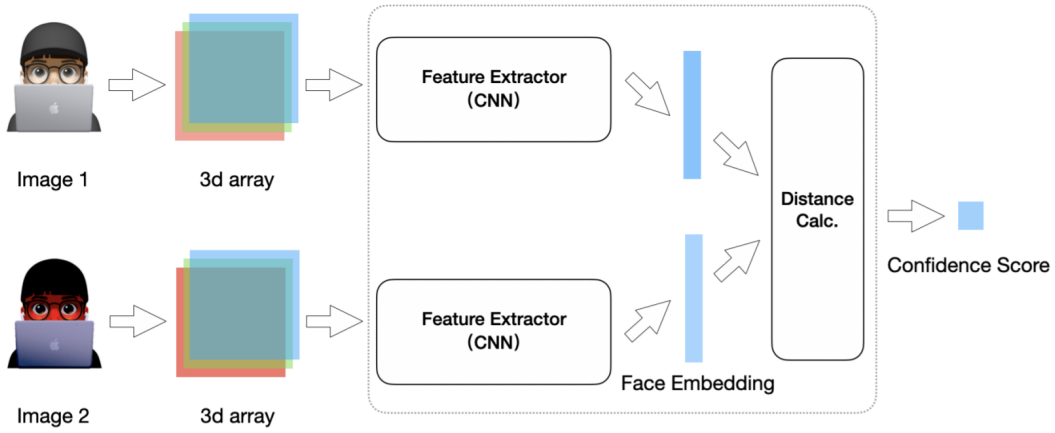


Figure 3: face verification architecture

5.2 Verification Problem Setup

The verification problem is to check if two images are of the same person or not, this is a complicated problem.

We will achieve this by using a feature extractor that is good at extracting discriminative features from the images, and then use the extracted feature vectors to compare the images.

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. We will use this property to compare the images and predict if they are of the same person or not.

[2] How close is close?

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

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 are unknown identities, the unknown identities have a one to one mapping with the known identities, and your job is to predict this mapping, so essentially for each unknown identity you will have to predict the corresponding known identity.

Hence, for this problem you will have find the similarity between each unknown identity and all the known identities, and then predict the one with the highest similarity. This operation is called one to many comparison, and it can be vectorized, using Pytorch's similarity functions, so you can easily get a N x N matrix of similarities, where N is the number of known and unknown identities, and then you can easily get the index of the highest similarity for each unknown identity.

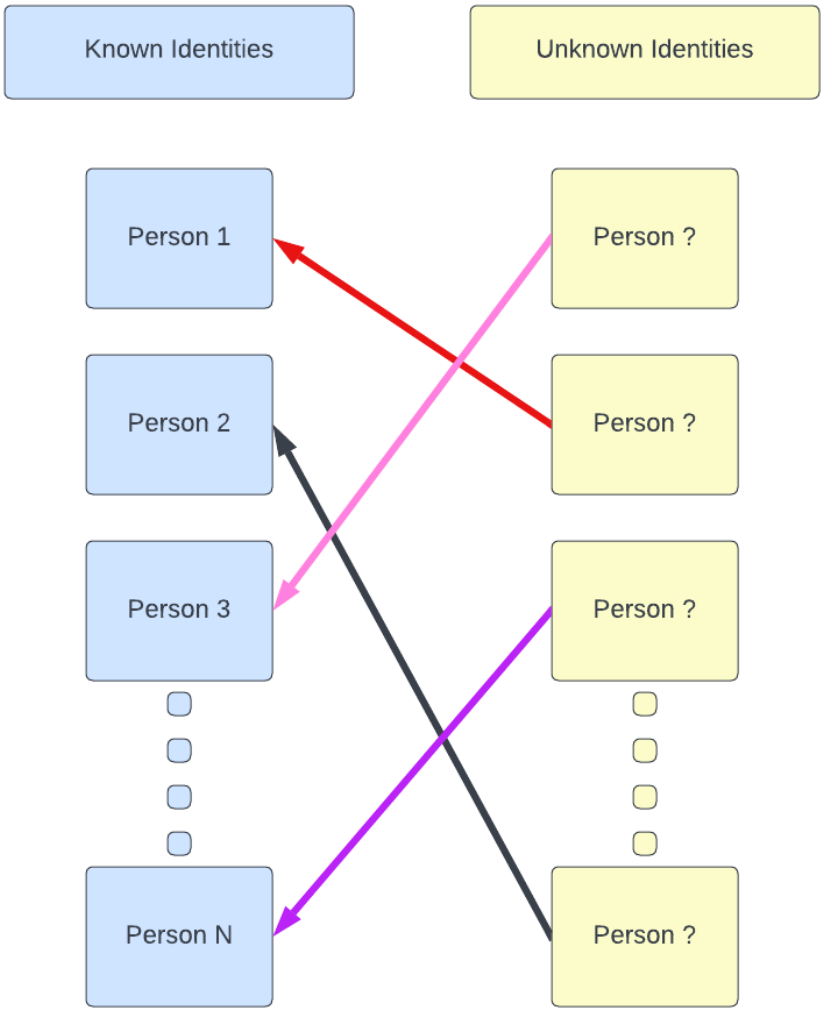


Figure 4: One to one mapping of the unknown identities to known identities

5.2.1 Take a step further

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], LM [2], L-GM[3], and other architectures such as Triplet Loss[13], SphereFace [4], CosFace[5], ArcFace[6] and UniformFace[7] to go beyond this.

Alternatively, you can remove the layer entirely and optimize the net using comparator-losses that optimize the network for the verification task, e.g., triplet-loss[8], pair wise loss[9].

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

5.3 Exploring other approaches

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

The second approach is actually called deep metric learning(DML): instead of modeling the classes, you are directly modeling the similarity between two images. The general goal is to make the minimum distance between negative pairs larger than the maximum distance between positive pairs [4].

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

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

Where d denotes Euclidean distance, and $y = 1/0$ indicates the pair P_i is positive/negative respectively. m is a margin. N denotes total number of training objectives.

There are two popular approaches to make pairs for your verification system. One is **offline selection**: pairs are generated before passed through the neural network. Another is **online selection**: pairs are generated in the mini-batch during training. For offline selection, please pay attention to the ratio of #negative pairs to #positive pairs. You are **advised** to set this ratio as 5:5, 6:4, 7:3. For online selection, one straightforward method is to select all $\frac{B(B \quad 1)}{2}$ pairs within a mini-batch of size B. You can also just select ***hard*** [5] pairs within the mini-batch, which is also referred to as **Hard Sample Mining**[11] [12].

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

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

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

wondering if there exists a Quintuplets, Sextuplets, Septuplets or even Octuplets loss, you can refer to the N-pair Loss [15], Lifted-Structure Loss [16], Softtriplet Loss [17] papers. It may also be possible for other advanced loss functions such as Pair-Wise Loss [18], Multi-Similarity(MS) [19], Mask Proxy(MP) [20] to give SOTA verification performance.

6 Evaluation System

6.1 Kaggle 1: Face Classification

This is quite straightforward,

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

6.2 Kaggle 2: Face Verification

This is also quite straightforward,

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

7 Submission

Following are the deliverables for this assignment:

- Kaggle submission for Face Classification.
- Kaggle submission for Face Verification.
- A one page write up describing your model architecture, loss function, hyper parameters, any other interesting detail led to your best result for the above two competitions. Please limit the write up to one page. The link for submitting the writeup will be posted later on Piazza/Autolab.

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 Cosine Similarity VS Euclidean Distance

You may struggle with selecting a proper distance metric for the verification task. The most two popular distance metrics used in verification are cosine similarity and Euclidean distance. We would tell you in that both two metrics are able to reach SOTA score, but at least you should get an intuition on how to choose one of them. The metric should be training-objective-specific, where training objective refers to the loss function. Let us start with revisiting Softmax cross entropy:

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

Where Y_i is the label of X_i. If you take a thorough look at this formula, you will find that the objective is to make the vector(embedding) X_i be closer to the vector W_{Y_i} and be far away from other vectors W_{Y_j}. Under this rule, the W_{Y_i} is actually the center of i-th class. Because you are performing dot product between the class center and the embedding, then each embedding would be similar to its center in the **Angular Space**, which could be illustrated in the following Figure. 3. So during verification, you are strongly suggested to apply cosine similarity rather than Euclidean distance to compute the similarity score.

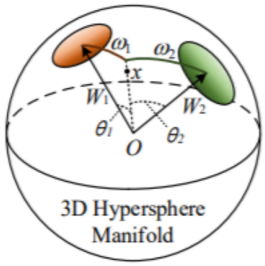


Figure 5: Angular Space [4]

Furthermore, if we design our own loss function e.g., in Eq. 3, you are suggested to apply Euclidean distance metric to compute similarity. (Is this RBF?)

$$Loss = \frac{1}{N}\sum_{i=1}^{N} log\frac{e^{jjW_{Y_i} \quad X_i jj^2}}{\sum_{j=1}^{N} e^{jjW_{Y_j} \quad X_i jj^2}} \tag{3}$$

Question left to you, what metric is probably better if you start with metric learning and apply the loss function in Eq. 1?

However, the aforementioned conclusions are not definitely true. We would tell you that sometimes Euclidean distance is also good when you apply softmax XE in Eq. 2 and cosine

similarity is also good when you apply Eq. 3 as loss function. We would just give you the following hint and let you explore it.

$$jjU \quad Vjj_2^2 = jjUjj_2^2 + jjVjj_2^2 \quad 2U^TV \tag{4}$$

Appendix B

B.1 B-way Classification

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

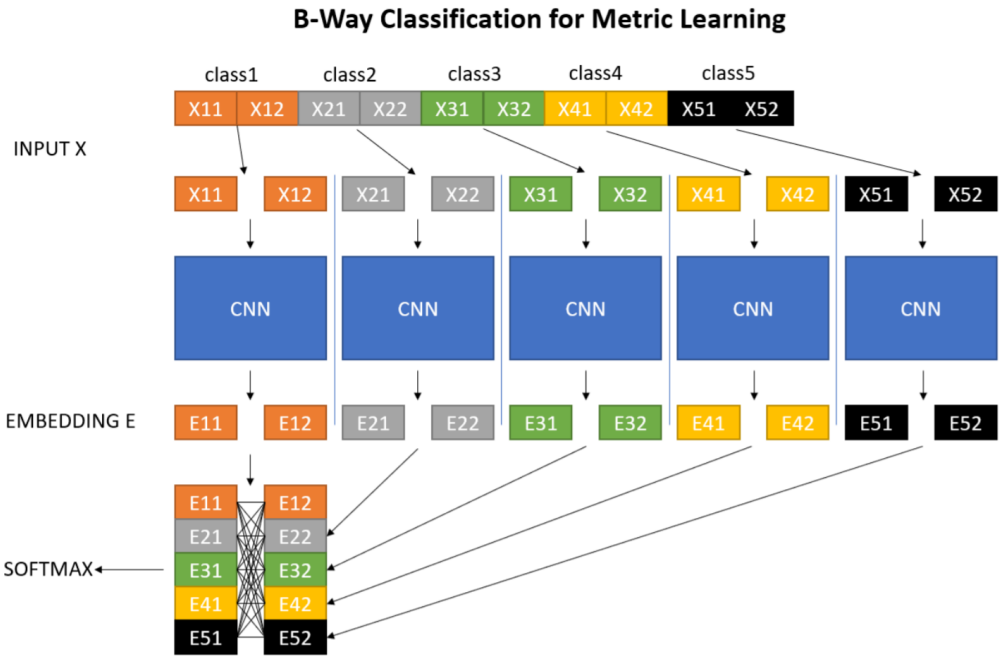


Figure 6: B-way classification for metric learning

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

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

This is actually called *Prototypical loss*, which is one of the SOTA metric learning losses currently and which is likely to give you a better AUC score than classification methods(Even

better than margin-based Softmax loss functions like CosFace/ArcFace). To apply this loss function, you may care about the following points:

- There is only one CNN backbone within a mini-batch though we present 5 in the example. (You can also apply Siamese Network)
- Label information is ignored when computing the loss objective. Labels are just 0,1,...B-1. Labels are only useful when building your dataset.
- You need to build a powerful dataset/dataloader to pass these B 2 data points into the network, which is the most pivotal part in the whole work
- $E_{i1}^T E_{i2}$ could be replaced by $a \quad E_{i1}^T E_{i2} + b$, in which a and b are learnable parameters. It would usually be better to normalize embeddings.
- There is no supervision signal in the loss objective unlike multi-class classification. (Is this unsupervised learning?)

Just feel free to go through this method!

References

[1] Yandong Wen, Kaipeng Zhang, Zhifeng Li, and Yu Qiao. A discriminative feature learning approach for deep face recognition. In European conference on computer vision, pages 499–515. Springer, 2016.

[2] Weiyang Liu, Yandong Wen, Zhiding Yu, and Meng Yang. Large-margin softmax loss for convolutional neural networks. ProC. Int. Conf. Mach. Learn., 12 2016.

[3] W. Wan, Y. Zhong, T. Li, and J. Chen. Rethinking feature distribution for loss functions in image classification. In 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 9117–9126, 2018.

[4] Weiyang Liu, Yandong Wen, Zhiding Yu, Ming Li, Bhiksha Raj, and Le Song. Sphereface: Deep hypersphere embedding for face recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 212–220, 2017

[5] H. Wang, Yitong Wang, Z. Zhou, Xing Ji, Zhifeng Li, Dihong Gong, Jingchao Zhou, and Wenyu Liu. Cosface: Large margin cosine loss for deep face recognition. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 5265–5274, 2018.

[6] Jiankang Deng, J. Guo, and S. Zafeiriou. Arcface: Additive angular margin loss for deep face recognition. 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 4685–4694, 2019.

[7] Y. Duan, J. Lu, and J. Zhou. Uniformface: Learning deep equidistributed representation for face recognition. In 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 3410–3419, 2019.

[8] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 815–823, 2015.

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

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

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

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

[13] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 815–823, 2015.

[14] Weihua Chen, Xiaotang Chen, Jianguo Zhang, and Kaiqi Huang. Beyond triplet loss: a deep quadruplet network for person re-identification. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 403–412, 2017.

[15] Kihyuk Sohn. Improved deep metric learning with multi-class n-pair loss objective. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, Advances in Neural Information Processing Systems 29, pages 1857–1865. Curran Associates, Inc., 2016.

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

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

[18] H. Dhamyal, T. Zhou, B. Raj, and R. Singh. Optimizing neural network embeddings using a pairwise loss for text-independent speaker verification. In 2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU), pages 742–748, 2019.

[19] Xun Wang, Xintong Han, Weiling Huang, Dengke Dong, and Matthew R. Scott. Multi-similarity loss with general pair weighting for deep metric learning. 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 5017–5025, 2019.

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