

MHLR: Moving Haar Learning Rate Scheduler for Large-scale Face Recognition Training with One GPU

Xueyuan Gong¹, Yain-whar Si², Zheng Zhang³, Xiaochen Yuan⁴, Ke Wang¹,
Xinyuan Zhang¹, Cong Lin¹ and Xiaoxiang Liu^{1*}

¹Jinan University

²University of Macau

³Harbin Institute of Technology, Shenzhen

⁴Macao Polytechnic University

xygong@jnu.edu.cn, fstasp@um.edu.mo, darrenzz219@gmail.com, xcyuan@mpu.edu.mo,
tlxx@jnu.edu.cn, wangke@jnu.edu.cn, zhangxy@jnu.edu.cn, conglin@jnu.edu.cn

Abstract

Face recognition (FR) has seen significant advancements due to the utilization of large-scale datasets. Training deep FR models on large-scale datasets with multiple GPUs is now a common practice. In fact, computing power has evolved into a foundational and indispensable resource in the area of deep learning. It is nearly impossible to train a deep FR model without holding adequate hardware resources. Recognizing this challenge, some FR approaches have started exploring ways to reduce the time complexity of the fully-connected layer in FR models. Unlike other approaches, this paper introduces a simple yet highly effective approach, Moving Haar Learning Rate (MHLR) scheduler, for scheduling the learning rate promptly and accurately in the training process. MHLR supports large-scale FR training with only one GPU, which is able to accelerate the model to 1/4 of its original training time without sacrificing more than 1% accuracy. More specifically, MHLR only needs 30 hours to train the model ResNet100 on the dataset WebFace12M containing more than 12M face images with 0.6M identities. Extensive experiments validate the efficiency and effectiveness of MHLR.

1 Introduction

Face recognition (FR) plays an important role in real-life applications. In recent years, the scale of FR training datasets have grown from 0.5M (Million) images with 0.01M IDs (Identities) in CASIA Webface [Yi *et al.*, 2014] to 42M images with 2M IDs in WebFace42M [Zhu *et al.*, 2021], which indicates the training of deep FR models have entered the large-scale datasets era, called large-scale FR training.

Deep FR models have been greatly advanced because of large-scale training datasets, such as MegaFace [Shlizerman *et al.*, 2016], MS1M [Guo *et al.*, 2016], and WebFace42M [Zhu *et al.*, 2021]. Yet, the basic requirement for training a deep FR model gradually grows from 1 node (Personal computer or server) with 4×GPUs to 1 node with

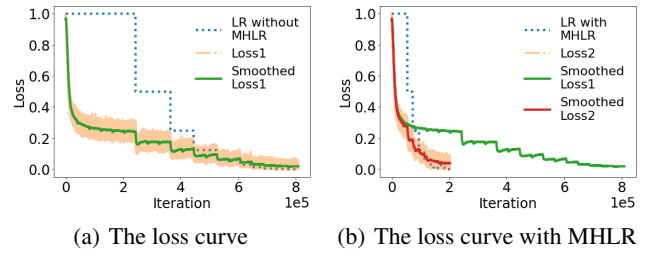


Figure 1: The loss curve of training ResNet100 on MS1MV3. All curves are normalized to $[0, 1]$ for putting in one figure. With MHLR, the loss after training $2e5$ steps is close to that after training $8e5$ steps without MHLR. Thus, only 1/4 of its original training time is required and we can train models with 1×GPU.

8×GPUs. The state-of-the-art FR approach, Partial FC (PFC) [An *et al.*, 2022], even employs at most 8 nodes with 8×GPUs for training a FR model, which is 64×GPUs in total. That shuts many researchers out the door of large-scale FR training. In fact, computing power has become a fundamental and critical resource in the deep learning area. It is nearly impossible to train a deep FR model without holding enough hardware resources, which can provide massive computing power.

Indeed, many cutting-edge FR approaches, e.g. PFC [An *et al.*, 2022], Faster Face Classification (F²C) [Wang *et al.*, 2022], Dynamic Class Queue (DCQ) [Li *et al.*, 2021a], and Virtual Fully-connected (Virtual FC) [Li *et al.*, 2021b], have already started to consider the problem of large-scale FR training. Nevertheless, those researches concentrate on the space and time complexity of the fully-connected layer in the model, which means large-scale FR training with multiple GPUs is still a basic requirement, in order to guarantee the feasible training time. Interestingly, the majority of researchers have rarely considered large-scale FR training on 1 node with only 1×GPU. Therefore, we would like to find out if it is possible to do large-scale FR training with 1×GPU while assuring its training time and accuracy.

Till now, the common practice of large-scale FR training is to train models for more than 20 epochs with at least

4×GPUs. In addition, a learning rate (LR) scheduler is employed for stabilizing the convergence of models. However, we identified that the loss curve would be shaped like a staircase while doing multiple step LR scheduling, as shown in Fig. 1(a). This phenomenon indicates that manual LR scheduling is vulnerable to training time waste, since many subsequences in the loss curve keep stationary. It is safe to schedule the LR early once the loss curve goes stationary. Thus, the intuition is to eliminate those stationary subsequences of the loss curve efficiently and effectively, as shown in Fig. 1(b). In this way, the deep FR model can converge faster without sacrificing much accuracy.

In this paper, we propose a simple yet effective approach, Moving Haar Learning Rate (MHLR) scheduler, which schedules the learning rate promptly and accurately. MHLR is able to guarantee the loss curve does not contain long stationary subsequences, in order to reduce the training time of the model. Thus, the required time for large-scale FR training with 1×GPU can be similar to 8×GPUs. The contribution of this paper can be summarized as follows:

- **Efficient.** MHLR is able to train the model for 5 epochs on 1×GPU, which costs 1/4 of its original training time. To be specific, on WebFace12M, MHLR requires 30H (hours) compared to the model trained for 20 epochs on 8×GPUs, which requires around 24H.
- **Effective.** MHLR sacrifices negligible accuracy for reducing training epochs from 20 to 5. Specifically, for ResNet100 trained on WebFace12M, the accuracy of MHLR on the dataset IJB-C is 97.20 while PFC-0.3 is 97.58. For ResNet100 trained on WebFace42M, the accuracy of PFC-0.3 on IJB-C is 97.82. Thus, we conclude the accuracy loss of MHLR is less than 1%.
- **Impactful.** MHLR proves that large-scale FR training with 1×GPU is not a mission impossible. It gives the possibility for many researchers to enter the door of large-scale FR training without massive hardware resources.

2 Related Work

In this section, we mainly reviewed the FR approaches relating to PFC [An *et al.*, 2022], since it is one of the state-of-the-art approaches aiming at large-scale FR training.

Back in 2017, SphereFace [Liu *et al.*, 2017] had already started to train FR models with 4×GPUs. It first introduced mapping the face image onto a hyper-sphere in the feature space. CosFace [Wang *et al.*, 2018b] proposed to normalize both the feature vector and the class vector in order to stabilize the training process. It adopted 8×GPUs to train FR models. UniformFace [Duan *et al.*, 2019] introduced to learn equidistributed feature vectors in order to maximize the utilization of the feature space. It employed 4×GPUs to train FR models. ArcFace [Deng *et al.*, 2019a] proposed additive angular margin in order to maximize the inter-class separation and the intra-class compactness. It trained FR models with 4×GPUs. CurricularFace [Huang *et al.*, 2020] embedded the idea of curriculum learning into the loss function, in order to address easy samples in the early training stage

and hard ones in the later stage. It utilized 4×GPUs in the FR model training. MagFace [Meng *et al.*, 2021] suggested building a connection between the quality of images and the magnitude of their corresponding feature vectors, in order to improve the generalization ability of the model. It leveraged 8×GPUs in the FR model training. ElasticFace [Boutros *et al.*, 2022] proposed to allow flexibility in the push for inter-class separability, in order to give the decision boundary chances to extract and retract to allow space for flexible class separability learning. It used 4×GPUs to train FR models. Also, there are some papers in which we failed to find the hardware information, such as Large-Margin Softmax (L-Softmax) loss [Liu *et al.*, 2016], Additive Margin Softmax (AM-Softmax) loss [Wang *et al.*, 2018a], Circle loss [Sun *et al.*, 2020], and AdaFace [Kim *et al.*, 2022]. To sum up, all FR approaches employ multiple GPUs for training FR models, except the hardware information of some approaches is not given. Therefore, we can conclude that training FR models with multiple GPUs is a common practice in this area.

Recently, some FR approaches have already started to consider large-scale FR training. Dynamic Class Queue (DCQ) [Li *et al.*, 2021a] proposed to dynamically select and generate class vectors instead of using them all in the training process. It adopted 8×GPUs to train FR models. Similar to DCQ, Faster Face Classification (F²C) [Wang *et al.*, 2022] introduced Dynamic Class Pool (DCP) to store and update class vectors dynamically. It employed 8×GPUs for the FR model training. Instead of maintaining a queue or a pool, Partial Fully-Connected (PFC) [An *et al.*, 2022] suggested randomly selecting negative class vectors. It leveraged at least 8×GPUs for the FR model training. Virtual Fully-connected (Virtual FC) [Li *et al.*, 2021b] has a similar goal to our approach, which is doing large-scale FR training with 1×GPU. Virtual FC focuses on reducing the space and time complexity of the FC layer, yet we focus on training epochs, which are different and have no conflict. In summary, all the mentioned approaches above concentrate on the FC layer. Yet in this paper, we aim at training epochs, which means our approach is compatible with the other approaches.

3 Problem Statement

In this section, we first revisit the large-scale FR training. After that, the problem in this paper is discussed.

A face image is defined as $\mathbf{x} \in \mathbb{R}^{N \times N \times C}$, where N is the size of the image and C is the channel of it. Thus, a deep FR model is defined as $\mathbf{f} = \mathcal{F}(\mathbf{x})$, where $\mathbf{f} \in \mathbb{R}^D$ is the feature vector extracted from $\mathbf{x}$. In order to train the model, the equation of combined margin loss function used in ArcFace and Partial FC is reviewed as follows:

$$\mathcal{L} = -\frac{1}{B} \sum_{i=1}^B \log \frac{e^{\mathcal{M}(\theta_{y_i, i})}}{e^{\mathcal{M}(\theta_{y_i, i})} + \sum_{j=1, j \neq y_i}^C e^{s \cos \theta_{j, i}}} \quad (1)$$

$$\mathcal{M}(\theta_{y_i, i}) = \cos(m_1 \theta_{y_i, i} + m_2) - m_3$$

where B denotes the batch size, C is the number of classes, $\mathcal{M}$ means the marginal function, m_1 stands for the multiplicative angular margin, m_2 represents the additive angular margin, m_3 denotes the additive cosine margin, s is the

scale factor controlling the radius of the hyper-sphere in feature space, $\theta_{j,i}$ represents the angle between the class vector $\mathbf{w}_j$ and the feature vector $\mathbf{f}_i$, $\mathbf{w}_j \in \mathbb{R}^D$ stands for the j -th class vector, and $\mathbf{f}_i$ denotes the feature vector of the i -th image belonging to the y_i -th class. Thus, $\theta_{y_i,i}$ can be calculated by the equation $\theta_{y_i,i} = \arccos(\mathbf{w}_{y_i} \mathbf{f}_i / \|\mathbf{w}_{y_i}\| \|\mathbf{f}_i\|)$, and $\mathcal{M}(\theta_{y_i,i})$ can be computed afterwards. Yet, since the time complexity of $\arccos(\cdot)$ is pretty high, the variant in Eq. (2) is a better option. Note that m_1 will change the frequency of the cos function and destroy its monotonicity. Thus, we set m_1 as 1 in this paper and thus it is treated as a constant.

$$\begin{aligned} \mathcal{M}(\theta_{y_i,i}) &= \cos(\theta_{y_i,i} + m_2) - m_3 \\ &= \sin m_2 \cos \theta_{y_i,i} - \cos m_2 \sqrt{1 - \cos^2 \theta_{y_i,i}} - m_3 \end{aligned} \quad (2)$$

where $m_1 = 1$, $\sin m_2$, $\cos m_2$ and m_3 are constants which can be pre-computed and stored, and $\cos \theta_{y_i,i} = \mathbf{w}_{y_i} \mathbf{f}_i / \|\mathbf{w}_{y_i}\| \|\mathbf{f}_i\|$.

To this end, we are able to train the model $\mathcal{F}$. Specifically, $\mathcal{F}$ is trained with E epochs, where each epoch contains U training iterations. Thus, $\mathcal{F}$ is updated $T = E \cdot U$ times in total, where the model and the loss after t -th updating are denoted $\mathcal{F}_t$ and $\mathcal{L}_t$ respectively. Formally, all $\mathcal{F}_t$ forms a sequence (time-series) $\mathbf{F} = \{\mathcal{F}_0, \mathcal{F}_1, \dots, \mathcal{F}_T\}$. Similarly, all $\mathcal{L}_t$ forms another sequence $\mathbf{L} = \{\mathcal{L}_0, \mathcal{L}_1, \dots, \mathcal{L}_T\}$. Hence, we wish to identify a sequence of the learning rate $\Gamma = \{\gamma_0, \gamma_1, \dots, \gamma_T\}$, in order to minimize the final loss $\mathcal{L}_T$ with as less T as possible.

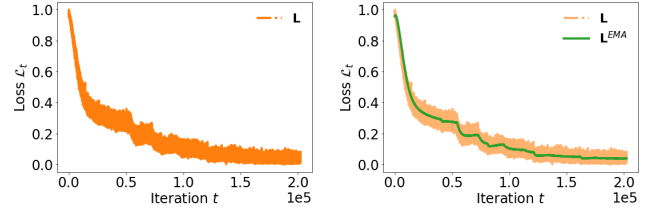
In practice, the learning rate (LR) λ is scheduled manually in the training process, which means the scheduling strategy is a hyper-parameter. The majority of FR approaches select Multiple Step LR (MultiStepLR) scheduler, where λ_t is reduced by a decay factor δ after certain training iterations. The new LR λ_{t+1} can be calculated by $\lambda_{t+1} = \lambda_t / \delta$. However, we discovered that MultiStepLR will always create many stationary subsequences in the loss curve, as already shown in Fig. 1(a). It represents that large-scale FR training suffers from training time waste, which indicates it is safe to schedule the LR early once the loss curve goes stationary. Therefore, we propose an easy yet effective LR scheduler, namely Moving Haar LR (MHLR) scheduler, in order to accelerate the model convergence.

4 The Proposed Approach

In this section, Moving Haar Learning Rate (MHLR) scheduler is introduced from three subsections. The first subsection presents the moving average part of MHLR. The second subsection demonstrates the Haar kernel part of MHLR. The third subsection summarizes the whole algorithm of MHLR.

4.1 Exponential Moving Average

Given a loss sequence $\mathbf{L} = \{\mathcal{L}_0, \mathcal{L}_1, \dots, \mathcal{L}_T\}$, it is hard to detect the inside stationary subsequences as it contains too many noises, which are caused by the optimization algorithm, Mini-batch Gradient Descent. More specifically, Batch Gradient Descent can generate a more smoothed loss sequence



(a) The loss curve $\mathbf{L}$ before smoothing (b) The loss curve $\mathbf{L}^{EMA}$ after smoothing

Figure 2: Smoothing the loss curve by EMA. As shown in (a), the original loss curve is hard to be analyzed since it contains too much noise. Thus, as shown in (b), we adopt EMA to smooth it. Note all curves are normalized

$\mathbf{L}$, since it considers all samples (face images) in every iteration. Thus, the loss function $\mathcal{L}$ remains unchanged. On the contrary, Mini-batch Gradient Descent considers distinct B samples in every iteration. Accordingly, the loss function $\mathcal{L}$ will always change in each iteration. This leads to the result that the generated loss sequence $\mathbf{L}$ is quite unstable, which contains too many noises, as shown in Fig. 2(a). Therefore, the first process in MHLR is smoothing the loss sequence $\mathbf{L}$, which filters noises in order to help the detection of stationary subsequences. The expected $\mathbf{L}$ after smoothing is shown in Fig. 2(b).

In this paper, we select Exponential Moving Average (EMA) to smooth $\mathbf{L}$. There are two reasons why we adopt EMA: 1) The loss sequence $\mathbf{L}$ is generated on-the-fly in the training process. EMA supports real-time updating, which is just compatible with our problem; 2) The time complexity of EMA in each iteration is only $\mathcal{O}(1)$, which indicates that it occupies negligible computations. The equation of EMA is given in Eq. (3).

$$\mathcal{L}_t^{EMA} = \begin{cases} \mathcal{L}_t, & t = 0 \\ \alpha \mathcal{L}_t + (1 - \alpha) \mathcal{L}_{t-1}^{EMA}, & t > 0 \end{cases} \quad (3)$$

where $\alpha \in [0, 1]$ indicates the importance of the current loss $\mathcal{L}_t$ compared to the average of previous losses $\mathcal{L}_{t-1}^{EMA}$, and $t \in \{0, 1, \dots, T\}$. Initially, when $t = 0$, $\mathcal{L}_0^{EMA} = \mathcal{L}_0$ directly. After that, $\mathcal{L}_t^{EMA}$ is calculated by the current loss $\mathcal{L}_t$ and the average of previous losses $\mathcal{L}_{t-1}^{EMA}$. In this paper, we α is set as 0.001.

Overall, EMA generates a smoothed loss sequence $\mathbf{L}^{EMA} = \{\mathcal{L}_0^{EMA}, \mathcal{L}_1^{EMA}, \dots, \mathcal{L}_T^{EMA}\}$ on-the-fly while generating the loss sequence $\mathbf{L}$, in order to support the detection of stationary subsequences.

4.2 Haar Convolutional Kernel

With EMA calculating $\mathbf{L}^{EMA}$ on-the-fly, we are able to detect stationary subsequences on it in real-time. Formally, given a smoothed loss sequence $\mathbf{L}^{EMA} = \{\mathcal{L}_0^{EMA}, \mathcal{L}_1^{EMA}, \dots, \mathcal{L}_T^{EMA}\}$, we wish to identify if the subsequence of $\mathbf{L}^{EMA}$ starting from S and ending at T , denoted $\mathbf{L}_{S,T}^{EMA} = \{\mathcal{L}_S^{EMA}, \mathcal{L}_{S+1}^{EMA}, \dots, \mathcal{L}_T^{EMA}\}$, keeps stationary statistically, where $0 \leq S < T$. Note $\mathcal{L}_T^{EMA}$ is always

the last value in $\mathbf{L}^{EMA}$ and it increases continuously, since $\mathbf{L}^{EMA}$ is updating in real-time.

In this paper, we adopt a simple Haar Convolutional Kernel (HCK) in the Haar-like Feature Descriptors to detect stationary subsequences. HCK is defined as $\mathcal{H}_{2s}$, where s represents the half size of it, the first half values of $\mathcal{H}_{2s}$ from 1 to s are -1 , the second half values of $\mathcal{H}_{2s}$ from $s+1$ to $2s$ are 1. For example, $\mathcal{H}_4 = \{-1, -1, 1, 1\}$. Thus, performing a convolutional operation on $\mathbf{L}^{EMA}$ using $\mathcal{H}_{2s}$, denoted $\mathcal{H}_{2s} * \mathbf{L}^{EMA}$, indicates calculating the difference between the sum of the first half values in $\mathbf{L}_{S,T}^{EMA}$ and the sum of the second half values in $\mathbf{L}_{S,T}^{EMA}$, where the size of $\mathbf{L}_{S,T}^{EMA}$ is $2s$. In this way, a stationary $\mathbf{L}_{S,T}^{EMA}$ will get a small difference value, while a $\mathbf{L}_{S,T}^{EMA}$ with a decreasing trend will get a relatively larger difference value. Hence, we are now able to distinguish the decreasing curve and the stationary curve. Nevertheless, a large $2s$ will cause two drawbacks:

1. The time complex of computing a single step convolution between $\mathcal{H}_{2s}$ and $\mathbf{L}_{S,T}^{EMA}$ is $\mathcal{O}(2s)$. It is inefficient to compute a long $\mathcal{H}_{2s}$ in every training iteration. For example, if $s = 5000$, a 10K times multiplication and a 10K times summation are needed in each iteration. In large-scale FR training, the total iterations T is commonly more than 500K, which results in an extra 10B (Billion) times computations in total.
2. A long $\mathcal{H}_{2s}$ requires a delay in detection. For instance, if $s = 5000$, then we have to wait until $\mathbf{L}^{EMA}$ accumulates 10K values before starting the detection, since the convolutional operation requires the size of $\mathbf{L}^{EMA}$ to be longer than $\mathcal{H}_{2s}$. However, we wish the stationary subsequence detection could start as soon as possible.

Thus, s is set to 1 in this paper, since it requires negligible computations and the shortest delay. Moreover, when $s = 1$, this convolutional operation degrades into calculating the difference $\mathcal{D}_t$ of $\mathcal{L}_{t-1}^{EMA}$ and $\mathcal{L}_t^{EMA}$. We denote the sequence of $\mathcal{D}_t$ as $\mathbf{D} = \{\mathcal{D}_1, \mathcal{D}_2, \dots, \mathcal{D}_T\}$ which can be calculated on-the-fly. The equation of it is given below:

$$\mathcal{D}_t = \mathcal{L}_{t-1}^{EMA} - \mathcal{L}_t^{EMA} \quad (4)$$

where $t \in \{1, 2, \dots, T\}$ here. Nonetheless, considering only two values in each step leads to a result that $\mathbf{D}$ is sensitive to noises. As shown in Fig. 3(a), it looks like a meaningless sequence, which is hard to use.

Therefore, we leverage EMA the second time, which aims at smoothing $\mathbf{D}$. The smoothed $\mathbf{D}$ is denoted as $\mathbf{D}^{EMA} = \{\mathcal{D}_1^{EMA}, \mathcal{D}_2^{EMA}, \dots, \mathcal{D}_T^{EMA}\}$. The equation of calculating $\mathbf{D}^{EMA}$ is shown as follows:

$$\mathcal{D}_t^{EMA} = \begin{cases} \mathcal{D}_t, & t = 1 \\ \beta \mathcal{D}_t + (1 - \beta) \mathcal{D}_{t-1}^{EMA}, & t > 1 \end{cases} \quad (5)$$

where $\beta \in [0, 1]$ indicates the importance of the current difference $\mathcal{D}_t$ compared to the average of previous differences $\mathcal{D}_{t-1}^{EMA}$. In this paper, β is set as 0.001.

Consequently, $\mathbf{D}^{EMA}$ is able to help us identify stationary subsequences, as shown in Fig. 3(b). In detail, we are able to observe a spike on $\mathbf{D}^{EMA}$ every time when there is a decline

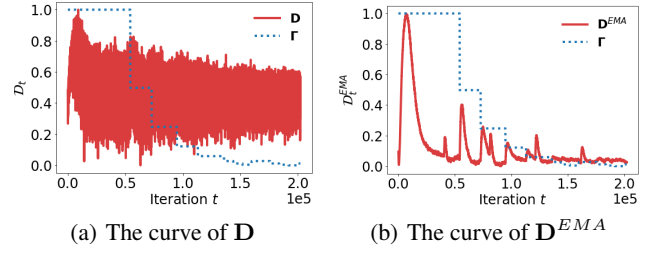


Figure 3: Examples of $\mathbf{D}$ and $\mathbf{D}^{EMA}$. As shown in (a), it is hard to find a relationship between $\mathbf{L}$ and $\mathbf{D}$. By contrast, as shown in (b), $\mathbf{D}$ is smoothed by EMA and denoted as $\mathbf{D}^{EMA}$. It is clear to observe a spike on $\mathbf{D}^{EMA}$ every time when there is a decline on $\mathbf{L}$. Note all curves are normalized

on $\mathbf{L}$, which indicates a rapid drop in the loss curve. Thus, the subsequence with small values before the spike in $\mathbf{D}^{EMA}$ represents the stationary part in $\mathbf{L}$. We conclude that MHLR can schedule $\mathcal{L}_t$ once identifying a consecutive subsequence of small $\mathcal{D}_t^{EMA}$.

4.3 Moving Haar Learning Rate Scheduler

To this end, we propose to combine EMA (Eq. (3)) and HCK (Eq. (4)) together. Thus, the calculation and storage for $\mathbf{L}^{EMA}$ are omitted. The combined approach is called Moving Haar Learning Rate (MHLR) scheduler. MHLR is able to detect the stationary subsequences on-the-fly in the training process. Its time and space complexity is $\mathcal{O}(1)$ in each iteration.

First, with Eq. (4) and Eq. (3), we are able to get a variant of the equation for calculating $\mathcal{D}_t$, as given in Eq. (6).

$$\begin{aligned} \mathcal{D}_t &= \mathcal{L}_{t-1}^{EMA} - \mathcal{L}_t^{EMA} \\ &= \mathcal{L}_{t-1}^{EMA} - \alpha \mathcal{L}_t - (1 - \alpha) \mathcal{L}_{t-1}^{EMA} \\ &= \alpha \mathcal{L}_{t-1}^{EMA} - \alpha \mathcal{L}_t \end{aligned} \quad (6)$$

Next, it is easy to have $\mathcal{D}_{t-1} = \alpha \mathcal{L}_{t-2}^{EMA} - \alpha \mathcal{L}_{t-1}$ from Eq. (6), where $\mathcal{D}_{t-1}$ just means one iteration before $\mathcal{D}_t$. After performing $\mathcal{D}_{t-1} - \mathcal{D}_t$, we have:

$$\begin{aligned} \mathcal{D}_{t-1} - \mathcal{D}_t &= \alpha \mathcal{L}_{t-2}^{EMA} - \alpha \mathcal{L}_{t-1} - \alpha \mathcal{L}_{t-1}^{EMA} + \alpha \mathcal{L}_t \\ &= \alpha (\mathcal{L}_{t-2}^{EMA} - \mathcal{L}_{t-1}^{EMA}) + \alpha (\mathcal{L}_t - \mathcal{L}_{t-1}) \\ &= \alpha \mathcal{D}_{t-1} + \alpha (\mathcal{L}_t - \mathcal{L}_{t-1}) \end{aligned} \quad (7)$$

From Eq. (6), we know $\mathcal{D}_1 = \alpha \mathcal{L}_0^{EMA} - \alpha \mathcal{L}_1$. Then, since $\mathcal{L}_0^{EMA} = \mathcal{L}_0$ in Eq. (3), we have $\mathcal{D}_1 = \alpha (\mathcal{L}_0 - \mathcal{L}_1)$. At last, we are able to get the final equation of $\mathcal{D}_t$ from Eq. (7), which is shown as follows:

$$\mathcal{D}_t = \begin{cases} \alpha (\mathcal{L}_{t-1} - \mathcal{L}_t), & t = 1 \\ (1 - \alpha) \mathcal{D}_{t-1} + \alpha (\mathcal{L}_{t-1} - \mathcal{L}_t), & t > 1 \end{cases} \quad (8)$$

To this end, calculating $\mathcal{D}_t$ by Eq. (8) does not include $\mathcal{L}_t^{EMA}$ anymore. In addition, it has three advantages: 1) $\mathcal{D}_t$ can be calculated on-the-fly. It requires only 1 step delay, i.e. when $t = 0$; 2) Its time complexity is $\mathcal{O}(1)$ in each iteration;

Algorithm 1 The algorithm of MHLR

Input: Previous learning rate γ_{t-1} , current loss $\mathcal{L}_t$, and previous loss $\mathcal{L}_{t-1}$

Parameter: The threshold λ , tolerance τ , $\alpha = 0.001$, and $\beta = 0.001$

Output: Current learning rate γ_t

Initialization: Iterations count $t = 1$, tolerance count $c = 0$, $\mathcal{D}_{t-1} = 0$, and $\mathcal{D}_{t-1}^{EMA} = 0$

```
1: if  $t = 1$  then
2:    $\mathcal{D}_t \leftarrow \alpha(\mathcal{L}_{t-1} - \mathcal{L}_t)$  ▷ Eq. (8)
3:    $\mathcal{D}_t^{EMA} \leftarrow \mathcal{D}_t$  ▷ Eq. (5)
4: else
5:    $\mathcal{D}_t \leftarrow (1 - \alpha)\mathcal{D}_{t-1} + \alpha(\mathcal{L}_{t-1} - \mathcal{L}_t)$  ▷ Eq. (8)
6:    $\mathcal{D}_t^{EMA} \leftarrow \beta\mathcal{D}_t + (1 - \beta)\mathcal{D}_{t-1}^{EMA}$  ▷ Eq. (5)
7: end if
8: if  $\mathcal{D}_t^{EMA} < \lambda$  then
9:   if  $c < \tau$  then
10:     $\gamma_t \leftarrow \gamma_{t-1}$ 
11:     $c \leftarrow c + 1$ 
12:   else
13:     $\gamma_t \leftarrow \gamma_{t-1}/2$ 
14:     $c \leftarrow 0$ 
15:   end if
16: end if
17:  $\mathcal{D}_{t-1} \leftarrow \mathcal{D}_t$ 
18:  $\mathcal{D}_{t-1}^{EMA} \leftarrow \mathcal{D}_t^{EMA}$ 
19:  $t \leftarrow t + 1$ 
20: return  $\gamma_t$ 
```

3) Its space complexity is also $\mathcal{O}(1)$, since it only needs to store the T -th and $(T - 1)$ -th values in sequences $\mathbf{D}$ and $\mathbf{L}$, instead of the whole sequences.

After calculating $\mathcal{D}_t$ by Eq. (8) and then $\mathcal{D}_t^{EMA}$ by Eq. (5), we set a threshold λ in order to determine if $\mathcal{D}_t^{EMA}$ is small enough, which indicates the loss sequence $\mathbf{L}$ keeps stationary. In addition, a tolerance τ is defined in case that MHLR is too sensitive to λ . Specifically, MHLR will reduce the learning rate γ_t if $\mathcal{D}_t^{EMA}$ keeps lower than λ , i.e. $\mathcal{D}_t^{EMA} < \lambda$, for τ iterations. After extensive experiments, we empirically set $\lambda = 5e - 5$ and $\tau = 0.05T$. The algorithm of MHLR is shown in Alg. 1.

5 Experiments

5.1 Implementation Details

Datasets. In this paper, we separately employ MS1MV2 [Deng *et al.*, 2019a], MS1MV3 [Deng *et al.*, 2019b], WebFace4M, WebFace8M, and WebFace12M [Zhu *et al.*, 2021] as the training sets. MS1MV2 is also called MS1M-ArcFace, which has more than 5.8M images with 0.085M identities. MS1MV3 is also called MS1M-RetinaFace, which contains more than 5M images with 0.09M IDs. WebFace42M [Zhu *et al.*, 2021] contains 42M images with 2M IDs. Afterwards, we randomly select 10%, 20%, and 30% samples from WebFace42M as WebFace4M, WebFace8M, and WebFace12M respectively. Hence, WebFace12M can have more than 12M images with 0.6M IDs in total. Note that the reason we do not employ WebFace42M

as our training set is that we have not applied PFC [An *et al.*, 2022] yet. Large-scale FR training without PFC will lead to a huge fully-connected layer, which is infeasible to have enough memory and time to train it on a single GPU.

Many public test sets are adopted to evaluate the performance of MHLR, including LFW [Huang *et al.*, 2007], CFP-FP [Sengupta *et al.*, 2016], AgeDB-30 [Moschoglou *et al.*, 2017], IJB-B [Whitelam *et al.*, 2017], and IJB-C [Maze *et al.*, 2018].

Experimental Settings. The experiments in the paper are implemented on a computer equipped with an Intel Core i9-13900K 3.00 GHz, 32 GB memory, and an NVIDIA GeForce RTX 4090 GPU. The operating system is Windows 11. In addition, the development environment is Anaconda 23.7.2 with Python 3.11.4 and Pytorch 2.0.1. Following the settings in recent papers [An *et al.*, 2022; Deng *et al.*, 2019a], input images are cropped and centered to the size of $112 \times 112 \times 3$. Each pixel in the image is normalized by subtracting 127.5 and then being divided by 128, which is mapped from $[0, 255]$ to $[-1, 1]$. Images are randomly flipped for data augmentation. Customized ResNet50 and ResNet100 [An *et al.*, 2022; He *et al.*, 2016] are employed as the backbones. The ResNet model outputs the feature vector of dimension 512. After that, feature vectors are input into the combined margin loss. For training set MS1MV3, $m_1 = 1$, $m_2 = 0.5$ and $m_3 = 0$. For training sets WebFace4M, WebFace8M, and WebFace12M, $m_1 = 1$, $m_2 = 0$ and $m_3 = 0.4$. The scale factor s is set as 64. In addition, the ResNet model is trained for 5 epochs on different training sets with only one single GeForce RTX 4090 GPU. The batch size is set as 128 samples in each updating. Stochastic Gradient Descent (SGD) with momentum 0.9 and weight decay $5e - 4$ is selected as the optimizer. The learning rate is set as 0.2, the decay factor $\delta = 2$, and it is automatically scheduled by MHLR. For EMA, α and β are both set as 0.001. The hyper-parameters in MHLR are set as $\lambda = 1e - 5$ and $\tau = 0.05$.

For testing, two feature vectors extracted from the original image and its horizontal-flipped one are summed together as the final feature vector. In addition, True Accept Rate (TAR) @ False Positive Rate (FAR) = $1e - 4$ is reported on IJB-B and IJB-C, denoted TAR@FAR = $1e - 4$.

5.2 Ablation Study

Different FR approaches with/without MHLR. In Tab. 1, we re-implement CosFace and ArcFace. After that, we train ResNet100 by them on different datasets with/without MHLR. The CosFace and ArcFace with MHLR are denoted as Cos+MHLR and Arc+MHLR. All approaches are trained on $1 \times \text{GPU}$. Without MHLR, ArcFace and CosFace need to train models for 20 epochs in common practice, which takes 36H (Hours) on MS1MV3 and 108H on WebFace12M. That represents a high time cost for researchers training a FR model. On the contrary, MHLR trains models for only 5 epochs, which takes 9H on MS1MV3 and 30H on WebFace12M. That is nearly $1/4$ of the training time required by ArcFace and CosFace. Moreover, on MS1MV3, the accuracy of ArcFace is 96.83 while that of Arc+MHLR is 96.41. On WebFace12M, the accuracy of CosFace is 97.55 while that of Cos+MHLR is 97.20. It means that the accuracy drop of

Method	Dataset	Epoch	Time	IJB-B	IJB-C
ArcFace	MS1MV3	20	36H	95.03	96.73
ArcFace	WebFace4M	20	32H	94.91	96.75
ArcFace	WebFace12M	20	107H	95.22	97.31
CosFace	MS1MV3	20	36H	95.02	96.54
CosFace	WebFace4M	20	32H	94.60	96.68
CosFace	WebFace12M	20	108H	95.41	97.55
Arc+MHLR	MS1MV3	5	9H	94.98	96.21
Arc+MHLR	WebFace4M	5	8H	94.85	96.17
Arc+MHLR	WebFace12M	5	30H	95.37	96.99
Cos+MHLR	MS1MV3	5	9H	94.91	96.11
Cos+MHLR	WebFace4M	5	8H	94.16	96.04
Cos+MHLR	WebFace12M	5	30H	95.46	97.08

Table 1: The performance of ArcFace and CosFace with/without MHLR. ResNet100 is selected as the model and trained with $1\times\text{GPU}$. $\text{TAR@FAR} = 1e-4$ is reported as the accuracy (%) on IJB-B and IJB-C

MHLR is tiny, which is less than 1%. Here, we would like to propose a question if it is worth to chase the small accuracy increasing while shutting the door of large-scale FR training for the majority of researchers. In summary, MHLR gives a competitive result with only 1/4 of the original training time. Note we have not re-implemented PFC yet, and thus it is infeasible to train models on WebFace42M with $1\times\text{GPU}$ since the limitation of the GPU memory.

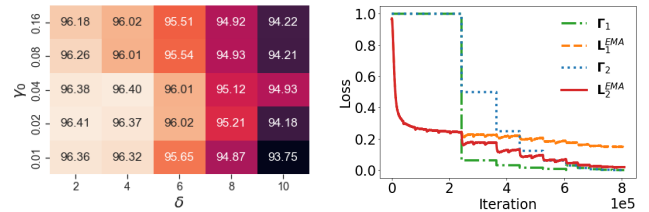
MHLR across different ResNet models In Tab. 2, we train different models on MS1MV3 and WebFace12M. All models are trained for 5 epochs on $1\times\text{GPU}$. Note the training on MV1MV3 is ArcFace+MHLR while that on WebFace12M is CosFace+MHLR. We witness a time increasing from ResNet18 all the way to ResNet200. More specifically, on MS1MV3, the training time and throughput are from 4H and 1876 images/s to 16H and 450 images/s respectively. On WebFace12M, the training time and the throughput are from 18H and 1015 images/s to 30H and 590 images/s respectively. Note we are not training ResNet200 on WebFace12M, since it is more than a NVIDIA Geforce RTX 4090 can handle. In addition, we observe a more significant training time increasing for ResNet50 and ResNet100. It indicates the computational bottleneck is not on the model for ResNet18, ResNet34, and ResNet50. By contrast, the computational time required by ResNet50 and ResNet100 overwhelms the computational time required by others. For instance, on MS1MV3, the training time of ResNet18 and ResNet34 do not have much difference if we do not arrange more threads in reading images, since the bottleneck is the speed of data loading. We can also observe that the throughput on WebFace12M is lower than that on MS1MV3. The reason is that the number of identities in WebFace12M is more than that in MS1MV3. Thus, it requires more computational time for the FC layer in the model. For accuracy, it is observed an increasing trend from ResNet 18 to ResNet200.

5.3 Hyperparameter Study

Initial learning rate γ_0 and decay factor δ In Fig. 4, we grid search the initial learning rate γ_0 and decay factor δ . All results are reported based on ResNet100 trained for 5 epochs

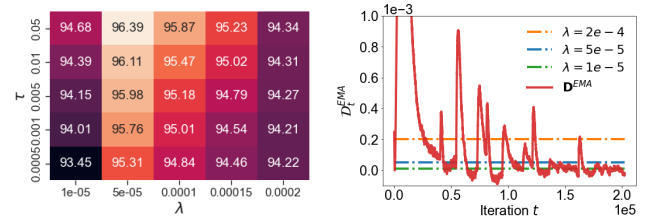
Model	Dataset	images/s	Time	IJB-B	IJB-C
ResNet18	MS1MV3	1876	4H	91.51	93.62
ResNet34	MS1MV3	1423	5H	93.63	95.31
ResNet50	MS1MV3	1203	6H	94.69	96.12
ResNet100	MS1MV3	786	9H	95.03	96.41
ResNet200	MS1MV3	450	16H	95.16	96.65
ResNet18	WebFace12M	1015	18H	92.13	94.43
ResNet34	WebFace12M	878	20H	94.12	95.87
ResNet50	WebFace12M	783	23H	95.25	96.85
ResNet100	WebFace12M	590	30H	95.68	97.20

Table 2: The performance of MHLR training different models. All models are trained for 5 epochs with $1\times\text{GPU}$. $\text{TAR@FAR} = 1e-4$ is reported as the accuracy (%) on IJB-B and IJB-C



(a) The heatmap of γ_0 and δ (b) The impact of decay factor δ

Figure 4: Grid searching for the initial learning rate γ_0 and decay factor δ . In (a), it shows the heatmap of γ_0 and δ , where $\text{TAR@FAR} = 1e-4$ on IJB-C is reported as the accuracy (%). In (b), it illustrates the impact of the decay factor δ , where $\delta = 8$ for Γ_1 and $\delta = 2$ for Γ_2 . Thus, Γ_1 declines fast and L_1^{EMA} goes stationary early



(a) The heatmap of λ and τ (b) Relationship of λ and D_t^{EMA}

Figure 5: Grid searching for the threshold λ and tolerance τ in MHLR. In (a), it shows the heatmap of $\text{TAR@FAR} = 1e-4$ on IJB-C is reported as the accuracy (%). In (b), it illustrates the relationship of different λ and D_t^{EMA} . The curve of D_t^{EMA} under λ will be treated as the signal to schedule the learning rate

on MS1MV3 with $1\times\text{GPU}$, while the test set is IJB-C. We observe an accuracy drop while δ is increasing, where a large δ represents the learning rate γ_t will decrease fast in the training process. That indicates a large δ is harmful for large-scale FR training, since the model will stop learning if γ_t becomes tiny in a very short time, as shown in Fig. 4(b). We also witness that $\gamma_0 = 0.02$ achieves the highest accuracy, since a small γ_t makes the learning stop while a big γ_t makes the learning unstable. In summary, we set $\gamma_0 = 0.02$ and $\delta = 2$ in this paper.

Method	Dataset	Model	#GPUs	Epoch	LFW	CFP-FP	AgeDB-30	IJB-B	IJB-C
					1 : 1 Verification Accuracy			TAR@FAR=1e-4	
CosFace [Wang <i>et al.</i> , 2018b]	Private	64-layer CNN	8	21	99.81	98.12	98.11	94.80	96.37
ArcFace [Deng <i>et al.</i> , 2019a]	MS1MV2	ResNet100	4	16	99.83	98.27	98.28	94.25	96.03
CurricularFace [Huang <i>et al.</i> , 2020]	MS1MV2	ResNet100	4	24	99.80	98.37	98.32	94.80	96.10
Sub-center [Deng <i>et al.</i> , 2020]	MS1MV3	ResNet100	8	24	99.80	98.80	98.31	94.94	96.28
MagFace [Meng <i>et al.</i> , 2021]	MS1MV2	ResNet100	8	25	99.83	98.46	98.17	94.08	95.97
ElasticFace [Boutros <i>et al.</i> , 2022]	MS1MV2	ResNet100	4	26	99.80	98.73	98.28	95.43	96.65
VPL [Deng <i>et al.</i> , 2021b]	Cleaned MS1M	ResNet100	8	20	99.83	99.11	98.60	95.56	96.76
F ² C [Wang <i>et al.</i> , 2022]	MS1MV2	ResNet50	8	20	99.50	98.46	97.83	-	94.91
F ² C [Wang <i>et al.</i> , 2022]	WebFace42M	ResNet100	8	20	99.83	99.33	98.33	-	97.31
Virtual FC [Li <i>et al.</i> , 2021b]	Cleaned MS1M	ResNet100	1	16	99.38	95.55	-	61.44	71.47
PFC-0.3 [An <i>et al.</i> , 2022]	WebFace4M	ResNet100	8	20	99.85	99.23	98.01	95.64	96.80
PFC-0.3 [An <i>et al.</i> , 2022]	WebFace12M	ResNet100	8	20	<u>99.83</u>	<u>99.40</u>	<u>98.53</u>	<u>96.31</u>	<u>97.58</u>
PFC-0.3 [An <i>et al.</i> , 2022]	WebFace42M	ResNet100	32	20	<u>99.85</u>	<u>99.40</u>	<u>98.60</u>	<u>96.47</u>	<u>97.82</u>
MHLR	MS1MV2	ResNet100	1	5	99.80	98.39	98.11	94.98	96.33
MHLR	MS1MV3	ResNet100	1	5	99.80	98.53	98.12	95.03	96.41
MHLR	WebFace4M	ResNet100	1	5	99.65	98.63	97.57	94.36	96.18
MHLR	WebFace8M	ResNet100	1	5	99.85	99.24	97.90	95.58	97.09
MHLR	WebFace12M	ResNet50	1	5	99.80	99.09	97.92	95.25	96.85
MHLR	WebFace12M	ResNet100	1	5	<u>99.83</u>	<u>99.17</u>	<u>98.02</u>	<u>95.68</u>	<u>97.20</u>

Table 3: Performance comparisons between MHLR and other state-of-the-art FR methods on various benchmarks. For LFW, CFP-FP, and AgeDB-30, 1:1 verification accuracy (%) is reported. For IJB-B and IJB-C, TAR@FAR= 1e-4 is reported as the accuracy (%).

The threshold λ and tolerance τ in MHLR In Fig. 5, we grid search the threshold λ and tolerance τ in MHLR. All results are reported based on ResNet100 trained for 5 epochs on MS1MV3 with 1×GPU, while the test set is IJB-C. We witness an accuracy decline while τ is decreasing, which indicates a small τ is too sensitive and hurried to schedule the learning rate. To the opposite, a big τ is slow in stationary subsequence detection. We also observe that $\lambda = 5e - 5$ has the highest accuracy. The reason is, as shown in Fig. 5(b), a small λ will filter the majority of $\mathcal{D}_t^{EMA}$ while a big λ considers too many $\mathcal{D}_t^{EMA}$ as the signal of scheduling the learning rate. In summary, we set $\lambda = 5e - 5$ and $\tau = 0.05$ in this paper.

5.4 Comparison with SOTA methods

In Tab. 3, we compared MHLR with other state-of-the-art (SOTA) methods. Other methods employ at least ×4 GPUs and averagely 20 epochs, while MHLR only uses ×1 GPU and 5 epochs. Thus, it requires only 1/4 of the training time compared to other methods. For example, it only requires 30H to train ResNet100 on WebFace12M with MHLR, as shown in Tab. 2. Comparing the accuracy of MHLR with other methods, its performance is competitive. For instance, on WebFace12M, the accuracy of MHLR is 97.20 while that of PFC-0.3 is 97.58. Yet, PFC-0.3 needs to be trained with 8×GPUs for 20 epochs, which is around 24 hours for training. Thus, we can conclude that it is meaningful doing large-scale training with 1×GPU.

In addition, PFC-0.3 and F²C also train ResNet100 on WebFace42M, which contains 42M images with 2M identities. The accuracy of PFC-0.3 and F²C on IJB-C are 97.82 and 97.31 respectively. In order to reach such a high performance, their training cost, including time and hardware resources, is huge. To be specific, it takes PFC nearly 25H to train ResNet100 on WebFace42M with 32×GPUs, and

it takes F²C days to train it on WebFace42M with only 8×GPUs. Nonetheless, MHLR is able to train ResNet100 on WebFace12M, not Webface42M, to get 97.20 on IJB-C by 30H with only 1×GPU. The accuracy gap is in 1%. Therefore, we conclude that large-scale FR training now faces the law of diminishing marginal utility, which means the cost increase rapidly in order to improve a small amount of the performance for FR models.

Please note Virtual FC also tries to train ResNet100 on self-cleaned MS1M with 1×GPU. Yet, it trains the model for 16 epochs, which requires more training time compared to MHLR for only 5 epochs. Besides, its accuracy on IJB-B and IJB-C has a large gap compared to other methods as reported in [An *et al.*, 2022]. By contrast, on MS1MV2 and MS1MV3, MHLR is competitive with other methods.

6 Conclusion

In this paper, we propose Moving Haar Learning Rate (MHLR) scheduler for large-scale face recognition training. MHLR schedules the learning rate promptly and accurately in order to reduce the converging time of FR models. As a result, MHLR is able to train the model with 1/4 of its original training time on 1×GPU by sacrificing less than 1% accuracy. We conclude that large-scale face recognition training now faces the law of diminishing marginal utility, which means the cost increase rapidly in order to improve a small amount of the performance for FR models.

Limitations. We have not implemented PFC yet, and thus training models on WebFace42M is still infeasible with 1×GPU. In the future, we will implement PFC+MHLR to validate the performance of trained models on WebFace42M with only 1×GPU. Afterwards, we will also conduct experiments on Masked Face Recognition (MFR) challenge [Deng *et al.*, 2021a].

Potential Societal Impacts. MHLR opens the gate of large-scale face recognition training to many researchers who do not have massive computing power, which is caused by lacking of expensive hardware sources. By attracting more researchers, this research direction will become thriving.

Acknowledgments

This work was supported in part by the Guangdong Basic and Applied Basic Research Foundation under Grant 2022A515110020, the Guangzhou Basic and Applied Basic Research Foundation under Grant 202201010476, the Fundamental Research Funds for the Central Universities under Grant 21621017, and the National Natural Science Foundation of China under Grant 62271232.

References

- [An *et al.*, 2022] Xiang An, Jiankang Deng, Jia Guo, Ziyong Feng, Xuhan Zhu, Jing Yang, and Tongliang Liu. Killing two birds with one stone: Efficient and robust training of face recognition cnns by partial FC. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*, pages 4032–4041, 2022.
- [Boutros *et al.*, 2022] Fadi Boutros, Naser Damer, Florian Kirchbuchner, and Arjan Kuijper. Elasticface: Elastic margin loss for deep face recognition. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, CVPRW*, pages 1577–1586, 2022.
- [Deng *et al.*, 2019a] Jiankang Deng, Jia Guo, Niannan Xue, and Stefanos Zafeiriou. Arcface: Additive angular margin loss for deep face recognition. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*, pages 4690–4699, 2019.
- [Deng *et al.*, 2019b] Jiankang Deng, Jia Guo, Debing Zhang, Yafeng Deng, Xiangju Lu, and Song Shi. Lightweight face recognition challenge. In *IEEE/CVF International Conference on Computer Vision Workshops, ICCVW*, pages 2638–2646, 2019.
- [Deng *et al.*, 2020] Jiankang Deng, Jia Guo, Tongliang Liu, Mingming Gong, and Stefanos Zafeiriou. Sub-center arcface: Boosting face recognition by large-scale noisy web faces. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *European Conference on Computer Vision, ECCV*, volume 12356, pages 741–757, 2020.
- [Deng *et al.*, 2021a] Jiankang Deng, Jia Guo, Xiang An, Zheng Zhu, and Stefanos Zafeiriou. Masked face recognition challenge: The insightface track report. In *IEEE/CVF International Conference on Computer Vision Workshops, ICCVW*, pages 1437–1444, 2021.
- [Deng *et al.*, 2021b] Jiankang Deng, Jia Guo, Jing Yang, Alexandros Lattas, and Stefanos Zafeiriou. Variational prototype learning for deep face recognition. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 11906–11915, 2021.
- [Duan *et al.*, 2019] Yueqi Duan, Jiwen Lu, and Jie Zhou. Uniformface: Learning deep equidistributed representation for face recognition. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*, pages 3415–3424, 2019.
- [Guo *et al.*, 2016] Yandong Guo, Lei Zhang, Yuxiao Hu, Xiaodong He, and Jianfeng Gao. Ms-celeb-1m: A dataset and benchmark for large-scale face recognition. In *European Conference on Computer Vision, ECCV*, volume 9907 of *Lecture Notes in Computer Science*, pages 87–102, 2016.
- [He *et al.*, 2016] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 770–778, 2016.
- [Huang *et al.*, 2007] Gary B Huang, Marwan Mattar, Tamara Berg, and Eric Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. In *University of Massachusetts, Technical Report*, 2007.
- [Huang *et al.*, 2020] Yuge Huang, Yuhan Wang, Ying Tai, Xiaoming Liu, Pengcheng Shen, Shaoxin Li, Jilin Li, and Feiyue Huang. Curricularface: Adaptive curriculum learning loss for deep face recognition. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*, pages 5900–5909, 2020.
- [Kim *et al.*, 2022] Minchul Kim, Anil K. Jain, and Xiaoming Liu. Adaface: Quality adaptive margin for face recognition. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*, pages 18729–18738, 2022.
- [Li *et al.*, 2021a] Bi Li, Teng Xi, Gang Zhang, Haocheng Feng, Junyu Han, Jingtuo Liu, Errui Ding, and Wenyu Liu. Dynamic class queue for large scale face recognition in the wild. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*, pages 3763–3772, 2021.
- [Li *et al.*, 2021b] Pengyu Li, Biao Wang, and Lei Zhang. Virtual fully-connected layer: Training a large-scale face recognition dataset with limited computational resources. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*, pages 13315–13324, 2021.
- [Liu *et al.*, 2016] Weiyang Liu, Yandong Wen, Zhiding Yu, and Meng Yang. Large-margin softmax loss for convolutional neural networks. In Maria-Florina Balcan and Kilian Q. Weinberger, editors, *International Conference on Machine Learning, ICML*, volume 48, pages 507–516, 2016.
- [Liu *et al.*, 2017] Weiyang Liu, Yandong Wen, Zhiding Yu, Ming Li, Bhiksha Raj, and Le Song. Sphreface: Deep hypersphere embedding for face recognition. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*, pages 6738–6746, 2017.
- [Maze *et al.*, 2018] Brianna Maze, Jocelyn C. Adams, James A. Duncan, Nathan D. Kalka, Tim Miller, Charles Otto, Anil K. Jain, W. Tyler Niggel, Janet Anderson, Jordan Cheney, and Patrick Grother. IARPA janus benchmark

- C: face dataset and protocol. In *International Conference on Biometrics, ICB*, pages 158–165, 2018.
- [Meng *et al.*, 2021] Qiang Meng, Shichao Zhao, Zhida Huang, and Feng Zhou. Magface: A universal representation for face recognition and quality assessment. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*, pages 14225–14234, 2021.
- [Moschoglou *et al.*, 2017] Stylianos Moschoglou, Athanasios Papaioannou, Christos Sagonas, Jiankang Deng, Irene Kotsia, and Stefanos Zafeiriou. Agedb: The first manually collected, in-the-wild age database. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPRW*, pages 1997–2005, 2017.
- [Sengupta *et al.*, 2016] Soumyadip Sengupta, Jun-Cheng Chen, Carlos Castillo, Vishal M Patel, Rama Chellappa, and David W Jacobs. Frontal to profile face verification in the wild. In *IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1–9, 2016.
- [Shlizerman *et al.*, 2016] Ira Kemelmacher Shlizerman, Steven M. Seitz, Daniel Miller, and Evan Brossard. The megaface benchmark: 1 million faces for recognition at scale. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*, pages 4873–4882, 2016.
- [Sun *et al.*, 2020] Yifan Sun, Changmao Cheng, Yuhan Zhang, Chi Zhang, Liang Zheng, Zhongdao Wang, and Yichen Wei. Circle loss: A unified perspective of pair similarity optimization. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*, pages 6397–6406, 2020.
- [Wang *et al.*, 2018a] Feng Wang, Weiyang Liu, Hanjun Dai, Haijun Liu, and Jian Cheng. Additive margin softmax for face verification. In *International Conference on Learning Representations Workshops, ICLRW*, 2018.
- [Wang *et al.*, 2018b] Hao Wang, Yitong Wang, Zheng Zhou, Xing Ji, Dihong Gong, Jingchao Zhou, Zhifeng Li, and Wei Liu. Cosface: Large margin cosine loss for deep face recognition. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*, pages 5265–5274, 2018.
- [Wang *et al.*, 2022] Kai Wang, Shuo Wang, Panpan Zhang, Zhipeng Zhou, Zheng Zhu, Xiaobo Wang, Xiaojiang Peng, Baigui Sun, Hao Li, and Yang You. An efficient training approach for very large scale face recognition. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*, pages 4073–4082, 2022.
- [Whitelam *et al.*, 2017] Cameron Whitelam, Emma Taborsky, Austin Blanton, Brianna Maze, Jocelyn C. Adams, Tim Miller, Nathan D. Kalka, Anil K. Jain, James A. Duncan, Kristen Allen, Jordan Cheney, and Patrick Grother. IARPA janus benchmark-b face dataset. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPRW*, pages 592–600, 2017.
- [Yi *et al.*, 2014] Dong Yi, Zhen Lei, Shengcai Liao, and Stan Z. Li. Learning face representation from scratch. *CoRR*, abs/1411.7923, 2014.
- [Zhu *et al.*, 2021] Zheng Zhu, Guan Huang, Jiankang Deng, Yun Ye, Junjie Huang, Xinze Chen, Jiagang Zhu, Tian Yang, Jiwen Lu, Dalong Du, and Jie Zhou. Webface260m: A benchmark unveiling the power of million-scale deep face recognition. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*, pages 10492–10502, 2021.