Partial FC: Training 10 Million Identities on a Single Machine

Xiang An[1,*], Xuhan Zhu[1,*], Yuan Gao[1,*], Yang Xiao[1], Yongle Zhao[1], Ziyong Feng[1]
Lan Wu[1], Bin Qin[1], Ming Zhang[1], Debing Zhang[1], Ying Fu[2]
[1]DeepGlint, [2]Beijing Institute of Technology

Abstract

Face recognition has been an active and vital topic among computer vision community for a long time. Previous researches mainly focus on loss functions used for facial feature extraction network, among which the improvements of softmax-based loss functions greatly promote the performance of face recognition. However, the contradiction between the drastically increasing number of face identities and the shortage of GPU memory is gradually becoming irreconcilable. In this work, we theoretically analyze the upper limit of model parallelism in face recognition in the first place. Then we propose a load-balanced sparse distributed classification training method, Partial FC, which is capable of using a machine with only 8 Nvidia Tesla V100 GPUs to implement training on a face recognition data set with up to 29 million IDs. Furthermore, we are able to train on data set with 100 million IDs in 64 RTX2080Ti GPUs. We have verified the effectiveness of Partial FC in 8 mainstream face recognition trainsets, and find that Partial FC is effective in all face recognition training sets. The code of this paper has been made available at https://github.com/deepinsight/insightface/tree/master/recognition/partial_fc.

1. Introduction

Face recognition is playing an increasingly important role in modern life and has been widely used in residential security, face authentication [4, 18, 19, 6, 16, 15] and criminal investigation. The softmax loss and its variants [17, 4, 15, 10] are widely used as objectives for face recognition. In general, they make global feature-to-class comparisons during the multiplication between the embedding features and the linear transformation matrix. In spite of that, when there are huge number of identities in the training set, the cost of storage and calculation of the final linear matrix easily exceed the current GPU capabilities, resulting in failure to train.

In this work, we analyze the drawbacks of model parallelism, and then propose Partial FC, a sparse variant of model parallel architecture for training face recognition. Since random sampling results in the imbalance of distributed computing and storage, which affects the efficiency of training, we propose an effective solution that can solve this problem and is significantly better than dense training methods. We verify the method in 8 large-scale face recognition training sets, with Partial FC, we are able to use only 8 Nvidia Tesla V100 GPUs to implement training on a face recognition data set with up to 29 million IDs and use 8 machines with 64 Nvidia 2080Ti GPUs to implement training on data set with 100M IDs. We achieve on par performance with dense trained model in the ultra-large-scale face recognition training set in the academic field, while less training time is required.

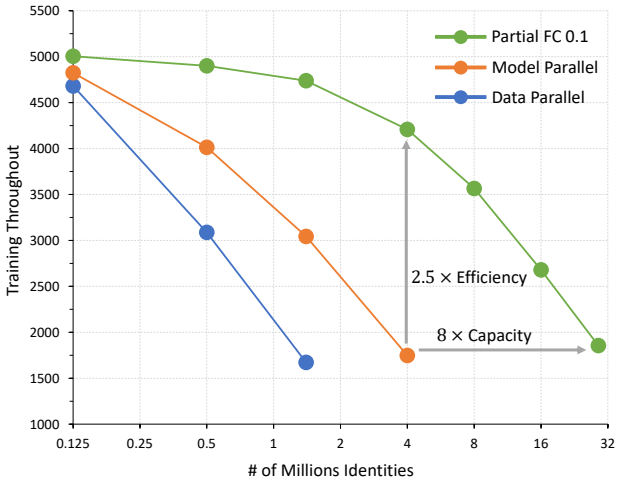

Figure 1: Partial FC increases training speed by 2.5 times on a 4-million-identity training set and increases maximum number of supported identities by 8 times.

2. Method

2.1. Model parallel

The most widely used classification loss function, softmax loss, can be described as

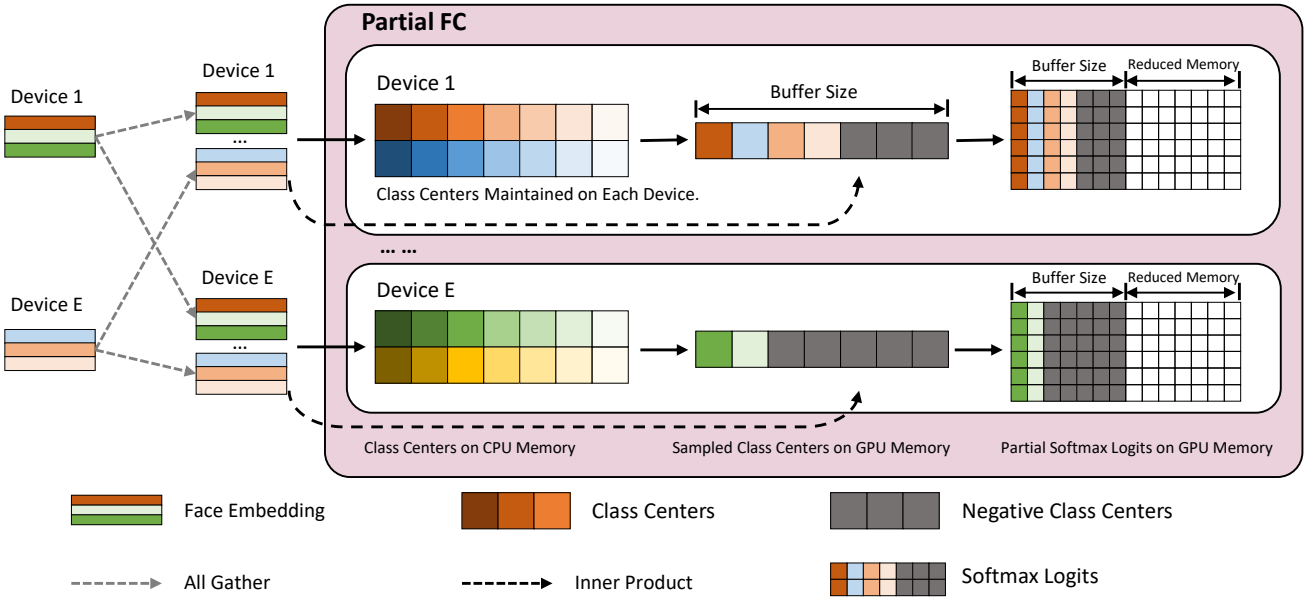

Figure 2: Class centers are represented by square, face embeddings are represented by rectangle, class centers are distributed on different GPUs. The embeddings of images and class centers have same color if they all represent the same identities. When the buffer is filled with positive class centers, the rest of the buffer is filled with random negative class centers, negative class centers are represented by gray. There is no overlap cross all GPUs, because the class centers they maintained are orthogonal. Partial FC reduces a large amount of GPU memory because it reduces both storage usage of the class centers and softmax logits.

$$L = -\frac{1}{N}\sum_{i=1}^{N} log\frac{e^{w_{y_i}^T x_i}}{\sum_{j=1}^{C} e^{w_j^T x_i}}, \tag{1}$$

where $x_i \in \mathbb{R}^d$ denotes the embedding feature of the i-th sample, belonging to the y_i-th class. $w_j \in \mathbb{R}^d$ denotes the j-th column of the weight $W \in \mathbb{R}^{d \times C}$. The batch size and the class number are N and C. Naturally, each column of the linear transformation matrix is viewed as a class center, and the j-th column of the matrix corresponds to the class center of class j. we denote w_{y_i} as positive class center of x_i, and the others are negative class centers. It is painful to train models with massive identities without using model parallel, subject to the memory capacity of a single GPU. The bottleneck exists in storing the matrix of softmax weight $W \in \mathbb{R}^{d \times C}$, where d denotes embedding feature dimension and C denotes the number of classes. A natural and straightforward approach to break the bottleneck is to partition W into k sub-matrices w of size $d \times \frac{C}{k}$ and places the i-th sub-matrices on the i-th GPU. Consequently, to calculate the final softmax outputs, each GPU has to gather features from all other GPUs, as the weights are split up among different GPUs. The definition of softmax function is

$$\sigma(X, i) = \frac{e^{w_i^T X}}{\sum_{j=1}^{C} e^{w_j^T X}}. \tag{2}$$

The calculation of the numerator can be done independently by each GPU as input feature X and corresponding weight sub-matrix w_i are stored locally. To calculate the denominator of the softmax function, sum of all $e^{w_j^T X}$ to be specific, information from all other GPUs have to be collected. Naturally, we can first calculate the local sum of each GPU, and then compute the global sum through communication. Compared with the naive data parallel, this implementation has negligible communication cost.

2.2. Memory Limits of Model Parallel

The model parallel can completely solve the storage and communication problems of w, since no matter how big C is, we can easily add more GPUs. So that each GPU's memory size storing sub-matrix w remains unchanged:

$$Mem_w = d \times \frac{C}{k} \times 4\ bytes. \tag{3}$$

However, w is not the only one stored on GPU memories. The storage of predicted logits suffers from the increase of total batch size. We denote the $logits$ storage on each GPU as $logits = Xw$, and then the memory consumption storing $logits$ on each GPU is therefore equal to:

$$Mem_{logits} = Nk \times \frac{C}{k} \times 4\ bytes, \tag{4}$$

where N is the mini-batch size on each GPU, and k is the number of GPUs. Assuming that the batch size of each GPU is constant, when C increases, in order to keep $\frac{C}{k}$ unchanged, we have to increase k at the same time. Hence, the GPU memory occupied by $logits$ will continue to increase, because the batch size of features increases synchronously with k. Suppose the mini-batch size on each GPU is 64 and the embedding feature dimension is 512, then 1 million classification task requires 8 GPUs and training 10 millions classification task requires at least 80 GPUs. We find that $logits$ will take up ten times as much memory cost as w, which makes storing logits new bottleneck to model parallel. The result shows that training tasks with massive identities cannot be solved by simply adding GPUs.

2.3. Partial FC

Sparse Softmax As mentioned above, model parallelism still has its limit. To solve this problem, we propose a possible training method. When computing the softmax loss function in each iteration, only a part of the class centers are randomly activated for computation. By doing so, a large amount of storage can be reduced, and a dense large model can be trained with a small amount of calculation. The class centers activated in each iteration are different, so that the parameters belong to all class centers will be updated throughout the training process. In order to efficiently select the class centers for computing in each iteration, we use this sampling method: positive class centers must be sampled, while negative class centers are randomly sampled, as illustrated in Figure 2.

Load Balancing Sampling Since the positive class centers of each batch are random, the number of positive class centers in each GPU is different, and because the negative class centers are selected randomly, both the positive centers and the negative centers are randomly distributed. This results in the imbalance of distributed computing and storage, which affects the efficiency of training. Hence, we further implement a load-balanced sampling method. In order to equalize the calculation and storage of each GPU, we set a memory buffer for each GPU. The size of the memory buffer is related to the sampling rate and the total number of classes. The positive center of each sample in the batch plays a critical role in the optimization of the neural network, therefore the positive centers must participate in model training and weights update. First, we load the memory buffer of each GPU with their respective positive class centers. In order to know which positive centers are in the batch, each GPU has to obtain the information of all samples in this batch. This information can be obtained through the collective communication primitive allgather, as depicted in Figure 2. After receiving the information of all samples in the batch, the class information of each sample will be checked. If the class of a sample is maintained by the current GPU, we transfer the weights corresponding to this class from CPU to GPU, otherwise we do not transfer. Due to the locality of model parallelism, each GPU activates only a fraction of all positive class centers of the overall batch, but for all GPUs, all positive class centers are activated. Afterwards, we randomly activate the remaining class centers maintained by each GPU to fill up the remaining memory buffer. By doing so, the class centers sampled in each GPU will not overlap each other, and the number of class centers involved in each GPU is equal. The subsequent training process is consistent with the normal model parallelism. When the weights in the GPU are updated, they are then transferred back to CPU.

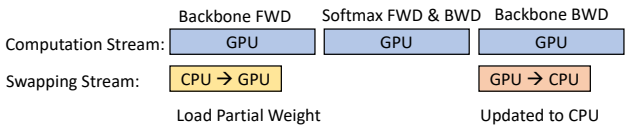

Figure 3: Sampling class centers and backbone computation in DNN training.

Efficiency As illustrated in Figure 3, the process of transferring weights from CPU to GPU is independent of the process of backbone forward, which means they can be done in parallel. Similarly, the process of transferring updated weights from GPU to CPU is independent of the process of backbone backward and thus they can be done in parallel too. Therefore, the IO overhead can be completely covered by calculations. When the number of classes is too large to be stored in GPU memory, we can store the weights in memory or even NVMe SSD without sacrificing the training speed. In this case, the GPU memory needed is decided by the sampling rate. One thing to be noticed, even if we store all weights on GPU, Partial FC still saves a large amount of GPU memory because it reduces the storage usage of final logits.

3. Experiment

3.1. Datasets and Settings

Datasets. Our training sets include CASIA[11], VGGface2[2], GlintAsia[20], MS1MV2[4], MS1MV3[5], Webface12M[22], and Megaface[9] . Furthermore, we clean Celeb-500k [1] and MS1M-Retinaface to merge into a new training set, which we call Glint360K. The released dataset contains 17 million images of 360K individuals. We explore efficient face verification datasets (*e.g.*, LFW [8], CFP-FP [14], AgeDB-30 [13]), IJB-C [12]) and ICCV21-MFR[3, 21] to test the robustness of Partial FC.

Method	ICCV21-MFR					AgeDB	CFPFP	LFW	IJB-C
	All	African	Caucasian	South Asian	East Asian				
CASIA	**36.79**	42.55	55.83	49.62	19.61	99.45	95.21	94.90	87.22
CASIA+pfc	**37.11**	38.93	53.82	48.67	19.93	99.37	95.43	94.60	84.97
VGGFace	**38.58**	35.26	54.30	44.08	24.10	99.55	97.41	95.08	91.22
VGGFace+pfc	**40.67**	36.77	60.18	49.04	24.26	99.68	98.53	95.40	92.49
GlintAsian	**62.66**	49.53	64.83	57.98	61.74	99.58	93.19	95.40	91.50
GlintAsian+pfc	**63.15**	50.37	65.23	57.94	61.82	99.65	93.03	95.23	91.14
MS1MV2	**77.70**	74.60	84.13	82.04	51.10	99.83	98.08	98.08	96.14
MS1MV2+pfc	**77.74**	74.73	84.88	82.80	52.51	99.78	98.07	98.02	96.08
MegafaceMS1M	**78.37**	74.14	82.25	77.22	60.20	99.75	97.56	97.40	95.35
MegafaceMS1M+pfc	**78.77**	73.69	82.95	78.79	57.57	99.80	97.87	97.73	95.40
MS1MV3	**82.52**	77.17	87.03	86.01	60.62	99.80	98.53	98.27	96.58
MS1MV3+pfc	**81.68**	78.13	87.29	85.54	58.93	99.80	98.44	98.17	96.43
Glint360K	**86.79**	84.75	91.41	90.09	66.17	99.82	99.14	98.45	97.13
Glint360K+pfc	**87.08**	85.27	91.62	90.54	66.81	99.82	99.14	98.45	97.02
Webface12m	**90.57**	89.36	94.18	92.36	73.85	99.80	99.20	98.10	97.12
Webface12m+pfc	**89.95**	89.30	94.02	92.38	73.01	99.82	99.14	98.12	97.01

Table 1: For ICCV2021-MFR, TAR is measured on 1:1 verification, with FAR less than 0.000001(1e-6). TAR@FAR=1e-4 is reported on the IJB-C datasets, 1:1 verification accuracy (%) is reported on the LFW, CFP-FP, AgeDB datasets. +pfc means using Partial FC training method.

Training Settings. We use ResNet50 [4, 7], as our backbone network. All experiments in this paper are implemented using Pytorch. The batch size is set to 1024 and models are trained on eight NVIDIA Tesla V100 (32GB) GPUs. we employ the SGD optimizer and the learning rate starts from 0.2. We set the feature scale s to 64 and cosine margin m of CosFace at 0.4, when Partial FC turns on, we set sampling rate to 0.1.

3.2. Robustness of Partial FC

In Table 1 we show this comparison on a wide variety of datasets, all of these are from celebrities. The number of identities in these training sets ranges from 8 thousands(VGGFace) to 0.75 millions(MegafaceMS1M), the number of faces in these training set ranges from 0.5 millions(CASIA) to 17millions(Glint360k). On average accuracy in eight data sets, Partial FC outperforms fully softmax baseline on ICCV-MFR-ALL and IJB-C by over 0.27 and -0.1, this can prove that Partial FC is competitive with fully softmax baseline in most data sets. Looking at individual datasets reveals some interesting behavior, Partial FC outperforms baseline on ICCV-MFR-ALL and IJB-C by over 2.1 and 1.9 on VGGFace2, we speculate this is due to VGGface2 dataset has a lot of noise, Partial FC effectively avoids model overfit into the training set due to the sampling of negative class center. Looking at where Partial FC notably underperforms, we see that Partial FC shows a little weak on several datasets on IJB-C, except MegafaceMS1M and VGGFace, this is because IJB-C's images are mainly derived from internet celebrities, it overlaps more or less with the mainstream celebrity training set, in general, better IJB-C results can be achieved as long as the training set is overfitted. Interestingly, there are a large number of non-celebrities in the MegafaceMS1M training set.

3.3. Performence of Partial FC

Training with Partial FC can save a lot of GPU memory and time because it reduces the computation of large scale classification. Using synthetic data, we test how many identities which Partial FC could train, we use IResnet50, set batch size to 512 and turn mixed precision training on. As shown in Figure 1, On a server with 8 32G V100 GPUs, the maximum number of identities supported by Partial FC is 29 millions, 8 times of model parallel. In a 4 millions ID training setting, Partial FC achieves 2.5 times faster than model parallel.

4. Conclusion

From a HPC and system perspective, we believe that Partial FC is a revolutionary innovation in large-scale classification tasks. Compared with model parallel, Partial FC can increase trainable identities by 8x and throughput by 2.5x. We verify the effectiveness of Partial FC in 8 mainstream face recognition training sets, which shows the significance of sparse training in face recognition. We hope this training method serves as a simple baseline for future research in large-scale facial recognition.

References

[1] Jiajiong Cao, Yingming Li, and Zhongfei Zhang. Celeb-500k: A large training dataset for face recognition. In *ICIP*, 2018.

[2] Qiong Cao, Li Shen, Weidi Xie, Omkar M Parkhi, and Andrew Zisserman. Vggface2: A dataset for recognising faces across pose and age. In *FG*, 2018.

[3] Jiankang Deng, Jia Guo, Xiang An, Zheng Zhu, and Stefanos Zafeiriou. Masked face recognition challenge: The insightface track report. In *Proceedings of the International Conference on Computer Vision Workshops*, 2021.

[4] Jiankang Deng, Jia Guo, Xue Niannan, and Stefanos Zafeiriou. Arcface: Additive angular margin loss for deep face recognition. In *CVPR*, 2019.

[5] Jiankang Deng, Jia Guo, Yuxiang Zhou, Jinke Yu, Irene Kotsia, and Stefanos Zafeiriou. Retinaface: Single-stage dense face localisation in the wild. *arXiv preprint arXiv:1905.00641*, 2019.

[6] Jiankang Deng, Yuxiang Zhou, and Stefanos Zafeiriou. Marginal loss for deep face recognition. In *CVPR Workshop*, 2017.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[8] Gary B. Huang, Marwan Mattar, Tamara Berg, and Eric Learned-Miller. Labeled faces in the wild: A database forstudying face recognition in unconstrained environments. *Workshop on Faces in 'Real-Life' Images: Detection, Alignment, and Recognition*, 2008.

[9] Ira Kemelmacher-Shlizerman, Steven M Seitz, Daniel Miller, and Evan Brossard. The megaface benchmark: 1 million faces for recognition at scale. In *CVPR*, 2016.

[10] Weiyang Liu, Yandong Wen, Zhiding Yu, Ming Li, Bhiksha Raj, and Le Song. Sphereface: Deep hypersphere embedding for face recognition. In *CVPR*, 2017.

[11] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *ICCV*, 2015.

[12] Brianna Maze, Jocelyn Adams, James A Duncan, Nathan Kalka, Tim Miller, Charles Otto, Anil K Jain, W Tyler Niggel, Janet Anderson, and Jordan Cheney. Iarpa janus benchmark–c: Face dataset and protocol. In *ICB*, 2018.

[13] Stylianos Moschoglou, Athanasios Papaioannou, Christos Sagonas, Jiankang Deng, Irene Kotsia, and Stefanos Zafeiriou. Agedb: The first manually collected in-the-wild age database. In *CVPR Workshop*, 2017.

[14] 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 *WACV*, 2016.

[15] Feng Wang, Weiyang Liu, Haijun Liu, and Jian Cheng. Additive margin softmax for face verification. *SPL*, 2018.

[16] Feng Wang, Xiang Xiang, Jian Cheng, and Alan L Yuille. Normface: l_2 hypersphere embedding for face verification. *arXiv:1704.06369*, 2017.

[17] 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.

[18] Hao Wang, Yitong Wang, Zheng Zhou, Xing Ji, Zhifeng Li, Dihong Gong, Jingchao Zhou, and Wei Liu. Cosface: Large margin cosine loss for deep face recognition. In *CVPR*, 2018.

[19] Yandong Wen, Kaipeng Zhang, Zhifeng Li, and Yu Qiao. A discriminative feature learning approach for deep face recognition. In *ECCV*, 2016.

[20] Debing Zhang. A distributed training solution for face recognition. *DeepGlint*, 2018.

[21] Zheng Zhu, Guan Huang, Jiankang Deng, Yun Ye, Junjie Huang, Xinze Chen, Jiagang Zhu, Tian Yang, Jia Guo, Jiwen Lu, Dalong Du, and Jie Zhou. Masked face recognition challenge: The WebFace260M track report. *arXiv:2108.07189*, 2021.

[22] Zheng Zhu, Guan Huang, Jiankang Deng, Yun Ye, Junjie Huang, Xinze Chen, Jiagang Zhu, Tian Yang, Jiwen Lu, Dalong Du, et al. Webface260m: A benchmark unveiling the power of million-scale deep face recognition. In *CVPR*, 2021.