J. Vis. Commun. Image R. 65 (2019) 102663

Contents lists available at ScienceDirect

J. Vis. Commun. Image R.

journal homepage: www.elsevier.com/locate/jvci

Benchmarking deep learning techniques for face recognition ☆

Qiangchang Wang, Guodong Guo *

West Virginia University, 109 Research Way – PO Box 6109, Morgantown, WV, USA

ARTICLE INFO

Article history:
Received 30 July 2019
Revised 23 September 2019
Accepted 29 September 2019
Available online 21 October 2019

Keywords:
Deep learning
Convolutional neural networks
Face recognition
GPU
PyTorch
TensorFlow
Caffe
AlexNet
ArcFace
Center-loss
CosFace
DenseNet
GoogLeNet
Inception-v3
LightCNN
ResNet
SphereFace
VGG

ABSTRACT

Recent progresses in Convolutional Neural Networks (CNNs) and GPUs have greatly advanced the state-of-the-art performance for face recognition. However, training CNNs for face recognition is complex and time-consuming. Multiple factors need to be considered: deep learning frameworks, GPU platforms, deep network models, training datasets and test datasets. The deep models under different frameworks may perform differently. Based on this concern, we compare three deep learning frameworks and benchmark the performance of different CNN models on five GPU platforms. The scalability issue is also explored. Our findings can help researchers select appropriate face recognition models, deep learning frameworks, GPU platforms, and training datasets for their face recognition tasks.

© 2019 Elsevier Inc. All rights reserved.

1. Introduction

During the past several decades, numerous face recognition approached are proposed [1–4 where various hand-crafted features are used. Recently, Convolutional Neural Networks (CNNs) greatly advance the face recognition performance. For example, the accuracy on the Labeled Faces in the Wild (LFW) dataset [5] has increased to 99.83% [6].

However, training deep neural networks is a very complex process. There are several factors that can affect the training process and evaluation performance. Firstly, there are a number of deep learning frameworks available, including Caffe from UC Berkeley [7], TensorFlow from Google [8] and PyTorch from Facebook [9], which are widely used frameworks among deep learning researchers [10]. However, these frameworks may have different data layouts (e.g. NCWH, NWHC) and convolution implementations (e.g. GEMM, FFT, Winograd), resulting in different performance.

Secondly, recent progress in deep CNNs has substantially improved state-of-the-art performance in face recognition, and there are a number of models available, such as AlexNet [11], VGG [12], GoogLeNet [13], ResNet [14], DenseNet [15], LightCNN [16], Incepetion-DenseNet [17], DDML [18], Center-loss [19], SphereFace [20], CosFace [21], UniformFace [22] and ArcFace [6]. The inherent structures in these models are somehow different. For example, GoogLeNet and DenseNet emphasize multi-scale feature learning; ResNet and DenseNet are deep networks to model complex data with a large spectrum of variations. Further, the developed deep models are usually reported on different training datasets, making it difficult to understand how different these models intrinsically, using the same training datasets.

Thirdly, training CNNs is a very time-consuming process. Graphical Processing Units (GPUs) are playing a key role in training networks. However, there are a number of GPU types, such as Titan Xp, GTX 1080Ti, Titan X(Pascal), Titan X(Maxwell) and Titan Z. It is

☆ This paper has been recommended for acceptance by Zicheng Liu.

* Corresponding author.
E-mail address: guodong.guo@mail.wvu.edu (G. Guo).

https://doi.org/10.1016/j.jvcir.2019.102663
1047-3203/© 2019 Elsevier Inc. All rights reserved.

interesting to know how differently these GPUs could perform in training deep models in practice.

Fourthly, there are different test datasets that may have different characteristics. In LFW dataset, almost all faces are frontal or close to frontal views; VGGFace2-test dataset [23] mainly focuses on pose and age variations; As for our processed IJB-A [24] quality dataset, it contains faces of different qualities; Disguise Faces in Wild (DFW) dataset [25] is the largest dataset of disguised faces and impostors; UMDFaces-test dataset [26] contains three tracks with different pose variations.

Fifthly, recent progress in face recognition is mainly being driven by advanced CNNs, fast GPUs, and large training datasets. Meanwhile, as CNNs are getting deeper, the requirement of large-scale training datasets becomes urgent. In recent years, several face datasets are made public with different scales, ranging from a few hundred thousand images, e.g., FaceScrub [27], CASIA-WebFace [28] and UMDFaces [26], to a few million images, e.g., VGGFace [29], MegaFace [30], MS-Celeb-1M [31] and VGGFace2 [23]. The deep CNNs may behave differently as the training datasets change. Many current face recognition models are trained on different training datasets, making it difficult to compare these models directly.

Given the variety of deep learning frameworks, face recognition models, GPU platforms, and training datasets, it is quite difficult for end users to select appropriate platforms to conduct their face recognition tasks. However, few works systematically investigate this issue. In [32], different types of neural networks are evaluated on a set of deep learning frameworks with different hardware platforms. However, there were no specific face recognition models or performance explored in [32]. Besides, more recent GPU types and newer versions of deep learning frameworks were not included in their study.

In this paper, based on several training datasets (i.e., UMDFaces, WebFace, MS-Celeb-1M and VGGFace2), we benchmark several face recognition models (i.e., AlexNet, VGG, GoogLeNet, ResNet, DenseNet, LightCNN, Center-loss, SphereFace, CosFace and ArcFace) under different deep learning frameworks (i.e., Caffe, TensorFlow and PyTorch) using different GPUs (i.e., Titan Xp, GTX 1080Ti, Titan X(Pascal), Titan X(Maxwell) and Titan Z) and test their performance on different datasets (i.e., LFW, VGGFace2-test, IJB-A quality, DFW challenge and UMDFaces-test).

The rest of paper is organized as follows. Section 2 introduces frameworks, deep models, and GPU platforms. Section 3 presents training and test datasets. Benchmarking results are presented in Section 4. Conclusion and future work are given in Section 5.

2. Frameworks, deep models, and GPU platforms

We briefly review three main deep learning frameworks, some popular deep learning face recognition models, and five widely used GPU platforms.

2.1. Deep learning frameworks

With the growing success of deep learning, there are many popular open-source deep learning frameworks which aim to help researchers quickly develop deep learning models, including Caffe, TensorFlow and PyTorch.

Caffe [7] is developed by Berkeley AI Research (BAIR) and GitHub community contributors. It enables researchers and practitioners to train and deploy general-purpose CNNs and other deep models efficiently on commodity architectures based on Python and MATLAB bindings.

TensorFlow [8] is designed by Google to operate at large scale and in heterogeneous environments. It uses dataflow graphs to represent both the computation in an algorithm and the state on which the algorithm operates.

PyTorch [9] is a scientific computing framework with two high-level features: tensor computation with strong GPU acceleration; deep neural networks built on a tape-based autodiff system. It aims to provide users with maximum flexibility and speed.

2.2. Deep models for face recognition

With the availability of large datasets, researchers have developed a number of CNN models. In this paper, we mainly focus on several publicly available models: AlexNet [11,33], VGG [12], GoogLeNet [13], Inception-v3 [34], ResNets [14], DenseNets [15], LightCNN [16], Center-loss [19], SphereFace [20], CosFace [21], and ArcFace [6]. The last five ones are specifically designed for face recognition, while others are originally developed for ImageNet classification [35].

The AlexNet-v1 [11] model is the first model that successfully applies CNNs for large-scale image classification. The Dropout technique is employed to reduce overfitting; Rectified Linear Units (ReLU) can prevent neurons from saturating; Local response normalization (LRN) layers aid generalization. In AlexNet-v2 [33], LRN layers are removed without remarkable performance change.

The VGG-16 [12] model is substantially deeper than the networks used before. 3×3 convolutional kernels are used to increase the network depth and reduce the number of parameters. In VGG-16-BN model, Batch Normalization (BN) layers [36] are added before ReLU, which can accelerate network training.

The GoogLeNet [13] model has much fewer parameters than the VGG-16, while still achieves a rather good performance. The Inception module is proposed to increase depth and width of deep networks. 1×1 convolutional kernels are applied to reduce the computational burden.

The Inception-v3 [34] model is one of the Inception families. It has high computational efficiency and low parameter number. Factorizing large convolutional filters is implemented by smaller and asymmetric convolutions. Efficient grid size reduction is proposed to reduce computational cost.

The ResNets [14] can be a depth of up to 152 layers but with a low complexity. A residual learning module is presented to ease the training of substantially deeper networks. ResNet-50 is a network with 50 layers.

The DenseNets [15] also have much deeper architectures. It connects each layer to every other layer in a feed-forward fashion. It can alleviate the vanishing-gradient problem, strengthen feature propagation, encourage feature reuse, and substantially reduce the number of parameters.

LightCNN [16] introduces Max-Feature-Map (MFM) to learn a robust face representation, which not only separates between noisy features and informative signals but also plays an important role in feature selection.

The Softmax loss function for LightCNN and ImageNet models is in the following:

$$L(I) = -\frac{1}{N}\sum_{i=1}^{N}\frac{e^{W_{y_i}^T x_i + b_{y_i}}}{\sum_{j=1}^{c} e^{W_j^T x_i + b_j}}, \tag{1}$$

where N and c denote the number of samples and classes, respectively. $x_i \in R^d$ refers the i^{th} deep feature, belonging to the y_i^{th} class. d is the feature dimension. $W_j \in R^d$ are the weights in the last fully connected layer connecting to the j^{th} class. b_j is the bias term of the j^{th} class.

The main drawback of Softmax loss is that it aims to separate different subjects well but does not explicitly minimize the intra-subject variations. To this aim, several loss functions are used to

enhance the discriminative power of the learned features. Center loss [19] is proposed to reduce intra-class variations. It can simultaneously learn a center for deep features of each class and penalize the distances between the deep features and their corresponding class centers. On one hand, center loss focuses on increasing intra-subject compactness. On the other hand, softmax loss aims at enlarging the inter-class distance. Intuitively, it is necessary to employ them jointly to supervise the training of CNNs. The formulation is in the following:

$$L(I) = -\sum_{i=1}^{N} \frac{e^{W_{y_i}^T x_i} + b_{y_i}}{\sum_{j=1}^{c} e^{W_j^T x_i + b_j}} + \frac{\lambda}{2}\sum_{i=1}^{N} \|x_i - c_{y_i}\|_2^2, \tag{2}$$

where the former part is the Softmax loss, and the later one is about Center loss. λ is used to balance the two loss functions. c_{y_i} is the y_i^{th} class center of deep features.

There exits another trend in the research community which increases the margin between different subjects in training. SphereFace [20] presents A-Softmax loss to impose discriminative constraints on a hypersphere manifold, which intrinsically matches the prior that faces lie on a manifold. Formally it optimizes:

$$L(I) = -\frac{1}{N}\sum_{i=1}^{N} log \frac{e^{\|x_i\|\psi(\theta_{y_i,i})}}{e^{\|x_i\|\psi(\theta_{y_i,i})} + \sum_{j\neq y_i} e^{\|x_i\| cos\theta_{j,i}}}, \tag{3}$$

where $\psi(\theta_{y_i,i}) = (-1)^k cos(m\theta_{y_i,i}) - 2k, \theta_{y_i,i} \in \left[\frac{k\pi}{m}, \frac{(k+1)\pi}{m}\right]$ and $k \in [0, m-1]$. $m \geqslant 1$ is an integer which controls the size of angular margin. $\theta_{j,i}$ is the angle between weight w_j and sample x_i.

In the CosFace loss, m is the cosine margin to maximize the decision margin in the angular space. The sample x_i is normalized and re-scaled to s.

$$L(I) = -\frac{1}{N}\sum_{i=1}^{N} log \frac{e^{s(\cos(\theta_{y_i,i})-m)}}{e^{s(\cos(\theta_{y_i,i})-m)} + \sum_{j\neq y_i} e^{s\cos\theta_{j,i}}}. \tag{4}$$

In the ArcFace loss, the additive angular margin penalty m is used to encourage the intra-class compactness and inter-class discrepancy.

$$L(I) = -\frac{1}{N}\sum_{i=1}^{N} log \frac{e^{s(\cos(\theta_{y_i,i}+m))}}{e^{s(\cos(\theta_{y_i,i}+m))} + \sum_{j\neq y_i} e^{s\cos\theta_{j,i}}}. \tag{5}$$

2.3. GPU platforms

Deep learning is a field with intense computational requirements, and the choice of GPUs can make big differences. In this work, three major GPU architectures are investigated. Furthermore, it is necessary to use multi-GPUs to speed up the training process and improve the model performance.

Table 1 gives some details of GPUs used in our experiments. Three different GPU architectures are benchmarked: Kepler (Titan Z), Maxwell (Titan X(Maxwell)) and Pascal (Titan X(Pascal), GTX 1080Ti and Titan Xp). Notice that we only use one of the two GK110 chips in Titan Z for the single GPU comparison.

3. Datasets

In this section, five test datasets with different characteristics and four popular training datasets are introduced.

3.1. Training datasets

Experiments prove that a large number of labeled faces can help CNNs learn about variations they need to handle in the prediction stage [37]. We compare four training datasets and explore their effects on several test datasets.

The web-collected CASIA-WebFace [28] has 494,414 images of 10,575 identities. The authors claim that not all faces are detected and annotated correctly. However, they think a small number of noisy images may make trained models more robust.

The UMDFaces dataset [26] contains 367,888 faces from 8277 identities. Humanly curated bounding boxes for faces are provided. The authors claim that it provides more pose variations than the popular WebFace dataset.

The VGGFace2 dataset [23] has 3,141,890 images and 8631 subjects. Human verified bounding boxes around faces are provided. It covers a large range of poses, ages, professions, and ethnicities.

The original MS-Celeb-1M dataset [31] contains too much noise. To get a high-quality dataset, DeepGlint (http://trillion-pairs.deepglint.com/) refined the dataset and made it publicly available. There are 86,878 subjects with 3.92 million aligned images.

3.2. Test datasets

The LFW dataset [5] can be viewed as a milestone dataset in which images are crawled from the Internet. It contains 13,233 images of 5749 subjects with many variations in illumination, expression, etc. Following the verification protocol of unrestricted with label outside data [38], 6000 pairs are evaluated.

The VGGFace2-test dataset [23] has 169,396 images from 500 celebrities. The test has two scenarios. Pose template: a template consists of five faces from the same subject with a consistent pose which can be frontal, three-quarter or profile view. Age template: a template consists of five faces from the same subject with either an apparent age below 34 (deemed young) or 34 and above (deemed mature). Each subject is represented by averaging the feature vector of all faces in each subject set. Then the similarity score is computed as the cosine similarity between feature vectors representing each subject.

Based on the method in [39], the quality value for each face image in the IJB-A dataset [24] can be assessed [40]. The dataset has 1543, 13,491, and 6196 images of 500, 483, and 489 subjects for high, middle and low qualities, respectively. The cross-quality matching performance on two scenarios is tested: low to high (Low2High) and middle to high (Middle2High). We evaluate the performance at different False Acceptance Rate (*FAR*) values.

The DFW dataset [25] is the largest dataset of disguised faces and impostors. There are 1000 subjects (400 in the training set

Table 1
Some details about various GPUs that we tested.

	Titan Z	Titan X (Maxwell)	Titan X (Pascal)	GTX 1080Ti	Titan Xp
Boost Clock (MHz)	876	1075	1531	1582	1582
CUDA Cores	5760	3072	3584	3584	3840
Memory Speed (Gbps)	7	7	10	11	11.4
Standard Memory Config (GB)	12	12	12	11	12
Memory Interface width (bit)	768	384	384	352	384
Memory Bandwidth (GB/s)	672	336.5	480	484	547.7
GPU Architecture	Kepler	Maxwell	Pascal	Pascal	Pascal

and 600 in the test set) and 11,155 images. Each subject in the test set has one genuine image, some validation images, several disguised images and a few images of impostors. Each pair in the test set is assigned a ground-truth label ('positive', 'negative' or 'do not care'). The task is to recognize disguised faces as the subject that they belong to and identify impostors. The performance is evaluated when $FAR = 1\%, 0.1\%$, respectively.

On top of a subset of the UMDFaces dataset [26], they developed a large testing protocol that contains three tracks: small pose variations (easy), medium pose variations (moderate) and large pose variations (difficult). There are 5000 positive pairs and 5000 negative pairs within each track. A large number of image pairs make it possible to compare performance at a low *FAR*.

4. Benchmarking experiments

We first discuss the implementation details in Section 4.1. Different face recognition models are compared in Section 4.2 wwhere model comparisons within the same deep learning framework and between different frameworks are discussed. Running time comparison of different models on various GPU platforms and scalability of different deep learning frameworks on multi-GPUs are presented in Section 4.3. Finally, Section 4.4 shows performance comparison of different training datasets on several test datasets.

4.1. Implementation details

For DFW and UMDFaces datasets, faces are cropped using the provided face coordinates and resized to the target image size. Other images are detected and cropped by the MTCNN [41] method. We use the Python implementation of the method MTCNN in [42]. Fig. 1 demonstrates the workflow on an example image. Referring to pre-processing operations in [43], we enlarge the bounding box size by 15% on each side and crop the image without the alignment procedure. If face detection is failed on one image, this image will be discarded if it is in the training dataset.

Deep models are trained on the WebFace dataset if not specified, so their relative performances can be compared. There are 453,413 detected faces in WebFace dataset. In order to reduce overfitting, all images are shuffled under PyTorch, Caffe and TensorFlow if the dataset is used for training. Let b, n refer to the batch size used in training and the image number in the dataset, respectively. There are $\lceil \frac{n}{b} \rceil$ batches in the dataset. Then a batch with b images is loaded sequentially from the beginning of the dataset, and then these images are pre-processed to feed the CNN model. Repeat these operations $\lceil \frac{n}{b} \rceil$ times until the end of the dataset. When the image number n is not divisible by the batch size b, the last batch size will be smaller than b. Under such a scenario in training, different frameworks have different strategies: the image pointer will seek to the start of the dataset recursively to get enough b images when training in Caffe; the last insufficient batch is directly used in PyTorch; TensorFlow will discard the smaller batch. During testing, the last insufficient batch will be processed in PyTorch, Caffe and TensorFlow. We input the face image and extract features from the penultimate layer for later analysis.

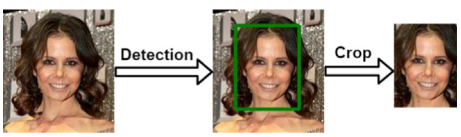

Fig. 1. The pre-processing on an example image, which consists of two stages, i.e. detection and crop stages. The detection stage is used to detect faces with the pre-trained MTCNN model, and the crop stage is to crop the face based on the bounding box.

4.2. Accuracy comparison of different face recognition models

Table 2 shows hyper-parameter settings and # of parameters of different face recognition models. After comparisons of different hyper-parameter settings on several models, there is little performance change compared with the original parameter settings in the proposed papers. Therefore, we decided that parameter settings of every model are the same as the paper in which the model was initially proposed. The momentum is 0.9. The weight-decay of the DenseNet-121, LightCNN-29, ResNet-50 models is 1e-4, GoogLeNet is 2e-4, Inception-v3 is 4e-4, and for other models, it is 5e-4. 2 GPUs are used to train the DenseNet-121, ResNet-50, VGG-16, VGG-16-BN, Center-loss and SphereFace models. 4 GPUs are employed to train the Softmax, CosFace and ArcFace based models.

We also compare the number of parameters in different frameworks. One can observe that the same face recognition model has similar number of parameters on three deep learning frameworks except the AlexNet-v2 and GoogLeNet. The AlexNet-v2 models in TensorFlow and PyTorch apply a zero padding of size 0 and 2 in the first convolutional layer, resulting in the last convolutional layer with size $256 \times 5 \times 5$ and $256 \times 6 \times 6$, separately. Because this layer is connected with a fully connected layer with units 4096, the PyTorch has $4096 \times 256 \times (6 \times 6 - 5 \times 5) = 11.5$ million more parameters compared with TensorFlow. As for the GoogLeNet model, the parameter number difference between Caffe and TensorFlow is that Caffe adds two auxiliary classifiers connected to intermediate layers, while TensorFlow does not.

We first compare model accuracy within the same deep learning framework and several loss functions in Section 4.2.1 and Section 4.2.2, respectively, and then illustrate model accuracy comparison between different frameworks in Section 4.2.3.

4.2.1. Model accuracy comparison within the same deep learning framework

Table 3 is about experimental results of PyTorch based models. For LFW dataset, the AlexNet-v2 model performs the worst, while others have similar performances. This is because the LFW dataset is a relatively easy dataset. For VGGFace2-test dataset, the VGG-16-BN and Inception-v3 models rank the first. For IJB-A quality dataset, the ResNet-50 model gets the first place on cross-quality face matching. This proves the fact that the network depth is important. VGG-16-BN achieves better performance than VGG-16. This is because BN layers act as a regularizer to improve the performance for the VGG-16. However, BN layers may fail for some models, and an appropriate batch size is needed to avoid the possible out-of-memory problem [44].

Experimental results about Caffe based models are shown in Table 4. For LFW dataset, GoogLeNet model can reach 97.8% accuracy. For VGGFace2-test and IJB-A datasets, LightCNN-29 model beats other models by a pronounced margin. It can be seen that LightCNN-29 can get better performance than other ImageNet based models. This proves that the effectiveness of Max-Feature-Map designed for the face recognition task.

Table 5 shows the performance of TensorFlow based models. For LFW dataset, Inception-v3 gets 98.5% accuracy, slightly better than other models. For VGGFace2-test dataset, GoogLeNet, Inception-v3 and DenseNet-121 are obviously better than others. For IJB-A quality dataset, GoogLeNet dramatically improves the matching accuracy across diverse qualities.

Among these models, results show that the GoogLeNet model is the most discriminative which gets 3 highest accuracies though

Table 2
The hyper-parameter settings and # of parameters in the models. The momentum is 0.9; iter means the global step during training; iterations per epoch (*ipe*) = # of images/(batch size).

Model	Batch size	Image size	Input size	Learning rate	Epochs or iters	Weight decay	# of params (million)		
							Caffe	TensorFlow	PyTorch
AlexNet-v1	256	256 × 256	227 × 227	$0.01 * 0.1^{(floor(iter/(21*ipe))}$	96	5e−4	100.17	–	–
AlexNet-v2	256	256 × 256	224 × 224	$0.01 * 0.1^{(floor(iter/(21*ipe)))}$	96	5e−4	–	89.53	100.33
DenseNet-121	40	256 × 256	224 × 224	$0.1 * 0.1^{(floor(iter/(10*ipe)))}$	30	1e−4	17.78	17.76	17.79
GoogLeNet	32	256 × 256	224 × 224	$0.01 * (1 - iter/(64 * ipe))^{0.5}$	64	2e−4	42.78	16.43	–
Inception-v3	25	299 × 299	299 × 299	$0.045 * 0.94^{(floor(iter/(2*ipe)))}$	100	4e−4	43.43	54.13	43.45
LightCNN-29	128	144 × 144	128 × 128	$0.1 * 0.457^{(floor(iter/(10*ipe)))}$	80	1e−4	8.19	–	–
ResNet-50	50	256 × 256	224 × 224	$0.1 * 0.1^{(floor(iter/(28*ipe)))}$	128	1e−4	45.16	45.18	45.18
VGG-16	128	256 × 256	224 × 224	$0.1 * 0.1^{(floor(iter/(17*ipe)))}$	74	5e−4	177.56	177.59	177.59
VGG-16-BN	128	256 × 256	224 × 224	$0.1 * 0.1^{(floor(iter/(17*ipe)))}$	74	5e−4	–	–	177.59
Softmax	512	112 × 112	112 × 112	$0.1 * 0.1^{(floor(iter/[16000,24000,28000]))}$	30,000	5e−4	–	–	–
Center-loss	256	112 × 96	112 × 96	$0.1 * 0.1^{(floor(iter/[16000,24000]))}$	28,000	5e−4	32.95	–	–
SphereFace	128	112 × 96	112 × 96	$0.1 * 0.1^{(floor(iter/[16000,24000]))}$	28,000	5e−4	68.45	–	–
CosFace	512	112 × 112	112 × 112	$0.1 * 0.1^{(floor(iter/[16000,24000,28000]))}$	30,000	5e−4	–	–	–
ArcFace	512	112 × 112	112 × 112	$0.1 * 0.1^{(floor(iter/[16000,24000,28000]))}$	32,000	5e−4	–	–	–

Table 3
Experimental results of various models in PyTorch.

PyTorch-based	LFW (%)	VGGFace2 (Similarity Score)						IJB-A (%)					
		Pose				Age		Low2High			Middle2High		
			Front	Three-quarter	Profile		Young	Mature	1%	0.1%	0.01%	1%	0.1%	0.01%
AlexNet-v2	93.1	Front	0.625	0.595	0.393	Young	0.534	0.231	29.1	11.9	5.1	55.3	34.9	23.0
		Three-quarter	0.599	0.651	0.510	Mature	0.259	0.606						
		Profile	0.390	0.508	0.625									
VGG-16	97.2		0.726	0.698	0.510		0.648	0.353	47.3	23.7	7.5	77.6	54.4	35.5
			0.705	0.739	0.603		0.382	0.711						
			0.517	0.604	0.695									
VGG-16-BN	97.6		**0.778**	**0.756**	**0.599**		**0.714**	0.431	51.5	28.0	7.9	81.3	60.0	38.7
			0.760	**0.789**	0.669		0.452	**0.761**						
			0.610	0.671	0.730									
Inception-v3	97.9		**0.778**	0.749	0.595		0.700	**0.454**	51.3	28.2	7.9	81.4	60.2	38.8
			0.756	**0.789**	**0.671**		**0.497**	0.758						
			0.606	**0.680**	**0.743**									
ResNet-50	**98.2**		0.756	0.726	0.556		0.669	0.412	**62.3**	**38.9**	**12.7**	**85.9**	**68.0**	**48.9**
			0.727	0.755	0.629		0.447	0.710						
			0.559	0.635	0.693									
DenseNet-121	97.8		0.755	0.731	0.576		0.682	0.419	58.8	33.1	9.8	83.3	62.4	42.1
			0.735	0.763	0.648		0.452	0.722						
			0.582	0.654	0.709									

unavailable in PyTorch because the LRN regularization is not supported. Despite only supported in Caffe, the LightCNN-29 model obtains 2 best results. The ResNet-50 and Inception-v3 models get 2 and 2 best results, respectively. The GoogLeNet and Inception-v3 models have Inception modules which consist of parallel convolutional kernels (1×1, 3×3 and 5×5). Inception modules allow to extract both local features (small convolutional kernels) and abstract features (larger convolutional kernels). The extracted multi-scale features can characterize face images at various levels, improving the face recognition performance. The ResNet-50 model uses identity mappings as bypassing paths, easing the training of very deep networks and improving the capacity to describe faces. On the other side, identity mappings connect features from two layers, which have different receptive field sizes. The Max-Feature-Map design allows the LightCNN-29 model to extract informative signals from noisy features.

4.2.2. Model accuracy comparison of several loss functions

Table 6 compares the Softmax loss function with several loss functions: Center-loss, SphereFace, CosFace and ArcFace.

In these models, SphereFace and ArcFace based models achieve the best accuracy on LFW dataset. Center-loss based model has the best accuracy on VGGFace2-test task. SphereFace based model outperforms others on the IJB-A quality task overall. It is observed that due to the complex face distributions, some loss functions may tend to perform well on some tasks, while have poor results on some specific tasks. Comprehensive investigation is necessary to achieve good results on specific tasks.

4.2.3. Model accuracy comparison between different frameworks

Tables 7–10 compare the VGG-16, Inception-v3, ResNet-50 and DenseNet-121 models across different deep learning frameworks. For the VGG-16 model, PyTorch gets the best accuracy on LFW and IJB-A quality datasets, and TensorFlow achieves the best result on VGGFace2-test dataset. For the Inception-v3 model, PyTorch gets the best accuracy on IJB-A quality dataset, and TensorFlow achieves the best result on LFW and VGGFace2-test datasets. For the ResNet-50 model, PyTorch can get the best results on LFW, VGGFace2-test and IJB-A quality datasets. For the DenseNet-121 model, PyTorch achieves the best performance on LFW and IJB-A quality datasets, and TensorFlow performs best on VGGFace2-test dataset.

Based on the above experiments, PyTorch based models tend to perform the best among these three frameworks, especially on

Table 4
Experimental results of various models in Caffe.

Caffe-based	LFW (%)	VGGFace2 (Similarity Score)						IJB-A (%), different FARs					
		Pose				Age		Low2High			Middle2High		
		Front	Three-quarter	Profile		Young	Mature	1%	0.1%	0.01%	1%	0.1%	0.01%
AlexNet-v1	94.6	Front	0.595	0.543	0.301	Young	0.477	0.259	12.1	4.2	1.7	35.8	21.3	13.2
		Three-quarter	0.557	0.609	0.432	Mature	0.281	0.541						
		Profile	0.310	0.430	0.583									
VGG-16	95.3		0.692	0.663	0.462		0.610	0.384	29.6	12.3	4.5	61.0	38.5	24.2
			0.675	0.709	0.564		0.395	0.656						
			0.476	0.565	0.681									
GoogLeNet	**97.8**		0.698	0.652	0.440		0.605	0.372	45.4	21.2	5.6	75.8	55.4	36.3
			0.660	0.701	0.540		0.392	0.658						
			0.450	0.541	0.644									
Inception-v3	96.1		0.712	0.692	0.560		0.626	0.415	29.65	12.34	4.52	61.2	38.53	24.24
			0.706	0.719	0.600		0.442	0.653						
			0.572	0.602	0.596									
ResNet-50	97.5		0.744	0.710	0.505		0.667	0.425	39.2	1.4	0.2	74.5	46.3	5.8
			0.718	0.753	0.591		0.455	0.697						
			0.513	0.594	0.677									
DenseNet-121	97.6		0.737	0.699	0.520		0.655	0.413	38.8	6.8	0.4	73.5	49.5	22.6
			0.707	0.746	0.604		0.442	0.698						
			0.533	0.611	0.690									
LightCNN-29	97.6		**0.764**	**0.740**	**0.600**		**0.691**	**0.472**	**50.4**	**28.2**	**14.0**	**76.2**	**55.9**	**36.7**
			0.748	**0.774**	**0.666**		**0.501**	**0.729**						
			0.607	**0.665**	**0.703**									

Table 5
Experimental results of various models in TensorFlow.

TensorFlow-based	LFW (%)	VGGFace2 (Similarity Score)						IJB-A (%), differnt FARs					
		Pose				Age		Low2High			Middle2High		
		Front	Three-quarter	Profile		Young	Mature	1%	0.1%	0.01%	1%	0.1%	0.01%
AlexNet-v2	91.1	Front	0.588	0.563	0.376	Young	0.497	0.255	25.2	11.5	5.9	47.9	31.4	20.7
		Three-quarter	0.567	0.608	0.470	Mature	0.262	0.563						
		Profile	0.380	0.473	0.574									
VGG-16	95.3		0.714	0.710	0.538		0.657	0.372	39.4	17.4	6.2	69.8	44.5	24.5
			0.713	0.751	0.624		0.386	0.721						
			0.545	0.621	0.708									
GoogLeNet	97.9		**0.829**	**0.811**	**0.683**		**0.783**	**0.543**	**58.3**	**34.1**	**15.6**	**84.1**	**62.8**	**37.9**
			0.816	**0.836**	**0.739**		**0.559**	**0.825**						
			0.693	**0.746**	**0.791**									
Inception-v3	**98.5**		0.814	0.790	0.652		0.752	0.512	47.7	25.6	9.7	76.6	53.4	31.6
			0.791	0.815	0.709		0.532	0.776						
			0.661	0.714	0.752									
ResNet-50	97.3		0.734	0.690	0.459		0.654	0.394	47.3	23.3	4.5	77.4	57.6	33.2
			0.695	0.726	0.557		0.423	0.687						
			0.464	0.562	0.654									
DenseNet-121	97.2		0.805	0.787	0.632		0.728	0.468	47.7	25.7	9.8	76.7	53.5	31.8
			0.791	0.811	0.700		0.488	0.790						
			0.646	0.707	0.766									

LFW and IJB-A quality datasets. TensorFlow based models can get excellent results on VGGFace2-test dataset. Caffe based models have the worst performance. These frameworks have diverse default settings, including data pre-processing steps as shown in Table 11 and weight initialization methods as indicted in Table 12, resulting in different performance as well. Diverse kernel initialization methods may have 1% effect on accuracy [45]. It is suggested that the PyTorch framework should be used in order to have better accuracy.

4.3. Running time comparison

Running time is a main factor that a user should take into consideration when training deep networks. These deep learning frameworks are benchmarked by using different batch sizes in 4 CNN models (i.e. VGG-16, Inception-v3, ResNet-50 and DenseNet-121). These four models have their characteristics to test the performance of frameworks. The below batch sizes are tried in every model: 16, 32, 64, 128, 256 and 512. However, some of them may fail because a too large batch size may lead to the out of memory problem.

The system configuration is Ubuntu 16.04.3 LTS. Hardwares include Intel(R) Core(TM) i7-6850 K CPU @ 3.60 GHz, 32 GB RAM and 512 GB SSD. Detailed information about several types of GPUs is shown in Table 1. Table 13 shows the information about different frameworks used in experiments. cuDNN [46] is a GPU-accelerated deep learning library, for neural network computing.

In subsequent experiments, running performance is evaluated by averaging 2000 iterations. The timing method used is shown in the following:

- Caffe: "Caffe time" function is used to calculate the average running time between two consecutive iterations. It reads original images from hard disk provided by a file list.

Table 6
Experimental results of several loss functions.

	LFW (%)	VGGFace2 (Similarity Score)						IJB-A (%), different FARs					
		Pose				Age		Low2High			Middle2High		
			Front	Three-quarter	Profile		Young	Mature	1%	0.1%	0.01%	1%	0.1%	0.01%
Softmax	99.0	Front	0.512	0.492	0.392	Young	0.477	0.419	55.3	**39.6**	20.4	81.3	72.7	53.6
		Three-quarter	0.510	0.517	0.412	Mature	0.417	0.404						
		Profile	0.406	0.411	0.397									
Center-loss	98.2		**0.857**	**0.846**	**0.751**		**0.816**	**0.606**	57.9	33.3	16.1	88.1	68.8	46.4
			0.847	**0.863**	**0.788**		**0.631**	**0.842**						
			0.752	**0.791**	**0.812**									
SphereFace	**99.2**		0.549	0.396	0.201		0.401	0.253	54.7	38.8	**22.2**	**90.3**	**80.0**	**64.3**
			0.398	0.498	0.334		0.249	0.428						
			0.217	0.332	0.487									
CosFace	99.1		0.552	0.539	0.427		0.531	0.468	**59.6**	31.8	0.4	81.9	78.4	20.9
			0.562	0.570	0.445		0.450	0.437						
			0.450	0.446	0.407									
ArcFace	**99.2**		0.580	0.571	0.462		0.559	0.501	**59.6**	0.7	0	82.9	77.6	0.3
			0.593	0.600	0.478		0.488	0.472						
			0.481	0.478	0.436									

Table 7
Experimental results of the VGG-16 model on Caffe, PyTorch and TensorFlow.

VGG-16	LFW (%)	VGGFace2 (Similarity Score)						IJB-A (%), different FARs					
		Pose				Age		Low2High			Middle2High		
			Front	Three-quarter	Profile		Young	Mature	1%	0.1%	0.01%	1%	0.1%	0.01%
Caffe-based	**97.6**	Front	**0.737**	0.699	0.520	Young	0.655	**0.413**	38.8	6.8	0.4	73.5	49.5	22.6
		Three-quarter	0.707	0.746	0.604	Mature	**0.442**	0.698						
		Profile	0.533	0.611	0.690									
PyTorch-based	97.2		0.726	0.698	0.510		0.648	0.353	**47.3**	**23.7**	**7.5**	**77.6**	**54.4**	**35.5**
			0.705	0.739	0.603		0.382	0.711						
			0.517	0.604	0.695									
TensorFlow-based	95.3		0.714	**0.710**	**0.538**		**0.657**	0.372	39.4	17.4	6.2	69.8	44.5	24.5
			0.713	**0.751**	**0.624**		0.386	**0.721**						
			0.545	**0.621**	**0.708**									

Table 8
Experimental results of the Inception-v3 model on Caffe, PyTorch and TensorFlow.

Inception-v3	LFW (%)	VGGFace2 (Similarity Score)						IJB-A (%), different FARs					
		Pose				Age		Low2High			Middle2High		
			Front	Three-quarter	Profile		Young	Mature	1%	0.1%	0.01%	1%	0.1%	0.01%
Caffe-based	96.1	Front	0.712	0.692	0.559	Young	0.626	0.415	29.7	12.3	4.5	61.2	38.5	24.2
		Three-quarter	0.706	0.719	0.597	Mature	0.442	0.653						
		Profile	0.572	0.602	0.596									
PyTorch-based	97.9		0.778	0.749	0.595		0.700	0.454	**51.3**	**28.2**	7.9	**81.4**	**60.2**	**38.8**
			0.756	0.789	0.671		0.497	0.758						
			0.606	0.680	0.743									
TensorFlow-based	**98.5**		**0.814**	**0.790**	**0.652**		**0.752**	**0.512**	47.7	25.6	**9.7**	76.6	53.4	31.6
			0.791	**0.815**	**0.709**		**0.532**	**0.776**						
			0.661	**0.714**	**0.752**									

Table 9
Experimental results of the ResNet-50 model on Caffe, PyTorch and TensorFlow.

ResNet-50	LFW (%)	VGGFace2 (Similarity Score)						IJB-A (%), different FARs					
		Pose				Age		Low2High			Middle2High		
			Front	Three-quarter	Profile		Young	Mature	1%	0.1%	0.01%	1%	0.1%	0.01%
Caffe-based	97.5	Front	0.744	0.710	0.505	Young	0.667	**0.425**	39.2	1.4	0.2	74.5	46.3	5.8
		Three-quarter	0.718	0.753	0.591	Mature	0.455	0.697						
		Profile	0.513	0.594	0.677									
PyTorch-based	**98.2**		**0.756**	**0.726**	**0.556**		**0.669**	0.412	**62.3**	**38.9**	**12.7**	**85.9**	**68**	**48.9**
			0.727	**0.755**	**0.629**		**0.447**	**0.710**						
			0.559	**0.635**	**0.693**									
TensorFlow-based	97.3		0.734	0.690	0.459		0.654	0.394	47.3	23.3	4.5	77.4	57.6	33.2
			0.695	0.726	0.557		0.423	0.687						
			0.464	0.562	0.654									

Table 10
Experimental results of the DenseNet-121 model on Caffe, PyTorch and TensorFlow.

DenseNet-121	LFW (%)	VGGFace2 (Similarity Score)						IJB-A (%), different FARs					
		Pose				Age		Low2High			Middle2High		
			Front	Three-quarter	Profile		Young	Mature	1%	0.1%	0.01%	1%	0.1%	0.01%
Caffe-based	97.6	Front	0.737	0.699	0.520	Young	0.655	0.413	38.8	6.8	0.4	73.5	49.5	22.6
		Three-quarter	0.707	0.746	0.604	Mature	0.442	0.698						
		Profile	0.533	0.611	0.690									
PyTorch-based	**97.8**		0.755	0.731	0.576		0.682	0.419	**58.8**	**33.1**	**9.8**	**83.3**	**62.4**	**42.1**
			0.735	0.763	0.648		0.452	0.722						
			0.582	0.654	0.709									
TensorFlow-based	97.2		**0.805**	**0.787**	**0.632**		**0.728**	**0.468**	47.7	25.7	9.7	76.7	53.5	31.6
			0.791	**0.811**	**0.700**		**0.488**	**0.790**						
			0.646	**0.707**	**0.767**									

Table 11
The data pre-processing of different CNN models in PyTorch/TensorFlow/Caffe.

Model name (Framework)	During training	During evaluation
AlexNet-v2, VGG-16, VGG-16-BN, ResNet-50, Inception-v3, DenseNet-121 (PyTorch)	(1) Crop the image to random size and aspect ratio, followed by the resizing operation. (2) Randomly flip the image horizontally. (3) Convert range of the image to [0, 1]. (4) Normalize the image with mean [0.485, 0.456, 0.406] and standard deviation [0.229, 0.224, 0.225].	(1) Center cropping and resize the image. (2) Convert the image pixel range to [0, 1]. (3) Normalize the image with mean [0.485, 0.456, 0.406] and standard deviation [0.229, 0.224, 0.225].
AlexNet-v2, VGG-16, ResNet-50, DenseNet-121 (TensorFlow)	(1) Get the smaller side of the image, followed by aspect-preserving resizing. (2) Randomly flip the image horizontally. (3) Randomly crop the image to 224×224. (4) Subtract means [123.68, 116.779, 103.939] from each image channel.	(1) Get the smaller side of the image, followed by aspect-preserving resizing. (2) Center cropping image to 224×224. (3) Subtract means [123.68, 116.779, 103.939] from each image channel.
GoogLeNet, Inception-v3 (TensorFlow)	(1) Convert the image pixel range to [0, 1]. (2) Generate cropped images using a randomly distorted bounding box and resize it. (3) Randomly distort the colors (brightness, hue, saturation or contrast). (4) Transform the image range to $[-1, 1]$.	(1) Convert the image pixel range to [0, 1]. (2) Center cropping and resize the image. (3) Transform the image range to $[-1, 1]$.
LightCNN-29 (Caffe)	(1) Randomly crop the image to the input size. (2) Randomly flip the image horizontally. (3) Transform dimensions. (4) Reorder image channels to BGR. (5) Subtract mean [127.5, 127.5, 127.5] and scale the image to range $[-1, 1]$.	(1) Center cropping and resize the image. (2) Transform dimensions. (3) Reorder image channels to BGR. (4) Subtract mean [87.1, 102.7, 134.6] and scale the image to range [0, 1].
AlexNet-v1, VGG-16, GoogLeNet, ResNet-50, Inception-v3, DenseNet-121 (Caffe)	(1) Randomly crop the image to the input size. (2) Randomly flip the image horizontally. (3) Transform dimensions. (4) Reorder image channels to BGR. (5) Subtract mean [87.1, 102.7, 134.6] and scale the image to range [0, 1].	

- PyTorch: the timing function in Python is used to calculate average iteration time. It directly reads original images from hard disks.
- TensorFlow: the internal timing function in the TensorFlow-Slim library [47] outputs time details in a specified number of iterations. It uses a processed file format called TFRecord.

It should be noted that these frameworks have very flexible programming APIs. It is possible that there exist other timing methods. Therefore, we should make it clear that our implementations are not necessarily the best approach for training.

4.3.1. Running time comparison of different frameworks and GPU platforms

The performance of the VGG-16 model is shown in the top left of Fig. 2. When the batch size is 128, the VGG-16 model fails to run on GTX 1080Ti. This is because GTX 1080Ti has a 11 GB memory space, compare to 12 GB in Titan Xp, Titan X(Pascal) and Titan X (Maxwell). Since only one of the two GK110 chips in Titan Z is used, there is 6 GB memory available, which is the reason why batch sizes 64 and 128 fail on Titan Z. On the other side, across different frameworks, PyTorch obtains the best performance among these three frameworks, followed by Caffe and then TensorFlow. For Titan Xp, the improvement of PyTorch compared with Caffe and TensorFlow is 26.5% and 45.1% when the batch size is 16, 31.5% and 40% when the batch size is 32, and 34.1% and 38.3% when batch size is 64. Besides, PyTorch is more memory-efficient since Caffe and TensorFlow have the out of memory problem under the batch size 128, 256 or 512.

The performance comparison of the Inception-v3 is shown in the top right of Fig. 2. Batch sizes 128, 256 and 512 fail, due to the out of memory issue. Across different GPU platforms, the Inception-v3 model displays relatively similar performance on 5 types of GPUs like the VGG-16 model. Across different frameworks, PyTorch obtains better performance than TensorFlow and Caffe under nearly each batch size. However, the speedup is different between PyTorch and TensorFlow when batch size changes. As for Titan Xp, when batch size is 16, PyTorch achieves about 1.16%

Table 12
The default weight initialization of diverse CNN models in Caffe/TensorFlow/PyTorch. Suppose parameters of one layer have shape $(output\_channel\_num, input\_channel\_num, kernel\_size, kernel\_size)$, $fan_{\text{in}} = input\_channel\_num * kernel\_size * kernel\_size$ and $fan_{\text{out}} = output\_channel\_num * kernel\_size * kernel\_size$.

Model name (framework)	Convolutional layers	Fully connected layers
VGG-16 (PyTorch)	$W \in N\left(0, \sqrt{\frac{2}{fan_{\text{out}}}}\right)$	$W \in \left[-\sqrt{\frac{1}{fan_{\text{in}}}}, \sqrt{\frac{1}{fan_{\text{in}}}}\right]$
VGG-16 (Caffe)	$W \in \left[-\sqrt{\frac{3}{fan_{\text{in}}}}, \sqrt{\frac{3}{fan_{\text{in}}}}\right]$	$W \in \left[-\sqrt{\frac{3}{fan_{\text{in}}}}, \sqrt{\frac{3}{fan_{\text{in}}}}\right]$
VGG-16 (TensorFlow)	$W \in \left[-\sqrt{\frac{6}{fan_{\text{in}}+fan_{\text{out}}}}, \sqrt{\frac{6}{fan_{\text{in}}+fan_{\text{out}}}}\right]$	-
Inception-v3 (PyTorch)	$W \in N(0, stddev), -2 \leqslant W \leqslant 2$	$W \in \left[-\sqrt{\frac{1}{fan_{\text{in}}}}, \sqrt{\frac{1}{fan_{\text{in}}}}\right]$
Inception-v3 (Caffe)	$W_{input} \in N\left(0, \sqrt{\frac{2}{fan_{\text{in}}+fan_{\text{out}}}}\right)$	$W_{input} \in N\left(0, \sqrt{\frac{2}{fan_{\text{in}}+fan_{\text{out}}}}\right)$
Inception-v3 (TensorFlow)	$W \in \left[-\sqrt{\frac{6}{fan_{\text{in}}+fan_{\text{out}}}}, \sqrt{\frac{6}{fan_{\text{in}}+fan_{\text{out}}}}\right]$	-
ResNet-50 (PyTorch)	$W \in N\left(0, \sqrt{\frac{2}{fan_{\text{out}}}}\right)$	$W \in \left[-\sqrt{\frac{1}{fan_{\text{in}}}}, \sqrt{\frac{1}{fan_{\text{in}}}}\right]$
ResNet-50 (Caffe)	$W_{input} \in N\left(0, \sqrt{\frac{2}{fan_{\text{out}}}}\right), W_{others} \in N\left(0, \sqrt{\frac{2}{fan_{\text{in}}}}\right)$	$W \in N(0, 0.01)$
ResNet-50 (TensorFlow)	$W \in \left[-\sqrt{\frac{6}{fan_{\text{in}}+fan_{\text{out}}}}, \sqrt{\frac{6}{fan_{\text{in}}+fan_{\text{out}}}}\right]$	-
DenseNet-121 (PyTorch)	$W \in N\left(0, \sqrt{\frac{2}{fan_{\text{in}}}}\right)$	$W \in \left[-\sqrt{\frac{1}{fan_{\text{in}}}}, \sqrt{\frac{1}{fan_{\text{in}}}}\right]$
DenseNet-121 (Caffe)	$W \in \left[-\sqrt{\frac{3}{fan_{\text{in}}}}, \sqrt{\frac{3}{fan_{\text{in}}}}\right]$	-
DenseNet-121 (TensorFlow)	$W \in \left[-\sqrt{\frac{6}{fan_{\text{in}}+fan_{\text{out}}}}, \sqrt{\frac{6}{fan_{\text{in}}+fan_{\text{out}}}}\right]$	$W \in \left[-\sqrt{\frac{6}{fan_{\text{in}}+fan_{\text{out}}}}, \sqrt{\frac{6}{fan_{\text{in}}+fan_{\text{out}}}}\right]$

speedup. However, when batch sizes are 32, 64 separately, the improvements are 20%, 27%. This shows that the larger the batch size is, the more obvious the speedup is. In addition, Caffe is less memory-efficient compared with TensorFlow and PyTorch because Caffe runs out of memory if the batch size is 32 or 64.

Table 13
Versions of deep learning frameworks used in experiments.

Frameworks	Major version	cuDNN	CUDA
Caffe	1.0.0	V7.0	V9.0
TensorFlow	1.6.0	V7.0	V9.0
PyTorch	0.3.1	V7.0	V9.0

The performance of the ResNet-50 model is shown in the bottom left of Fig. 2. There is out of memory problem if the batch size is 256 or 512. Across different frameworks, PyTorch achieves the best performance compared with TensorFlow and Caffe, except that the batch size is 64 on GTX 1080Ti. For the Titan Xp, speedup between PyTorch and TensorFlow is 25.4%, 28.2%, 31.5% and 27% when the batch size is 16, 32, 64 or 128 respectively. Titan Z and GTX 1080Ti cannot afford the batch size 128 under these three frameworks. Caffe based ResNet-50 model can run with a batch

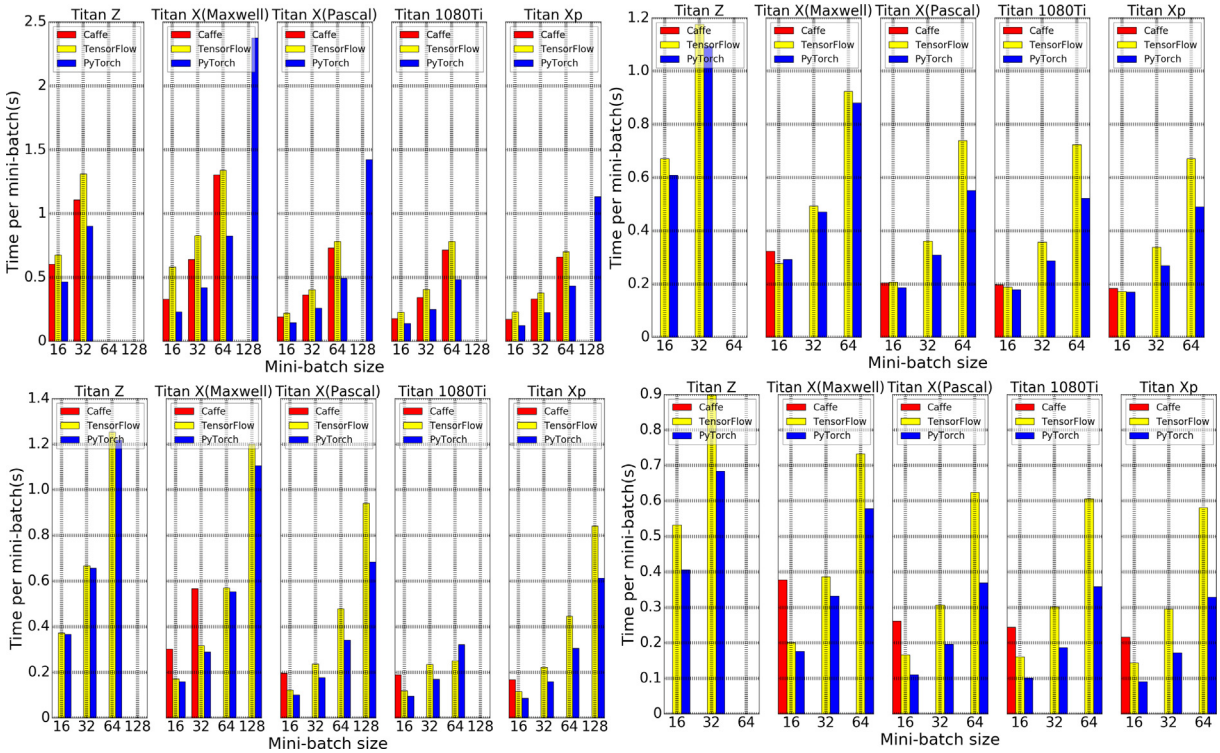

Fig. 2. Running time comparison of VGG-16 (top left), Inception-v3 (top right), ResNet-50 (bottom left), and DenseNet-121 (bottom right) models on GPU platforms.

size 16 on Titan X(Maxwell), Titan X(Pascal), GTX 1080Ti and Titan Xp. Since one of two chips on Titan Z has less memory, Caffe based ResNet-50 model fails when the batch size is 16.

The performance of the DenseNet-121 model is shown in the bottom right of Fig. 2. It fails when the batch size is 128, 256 or 512. PyToch is better than other frameworks. For Titan Z, batch sizes of PyTorch and TensorFlow are 16 and 32; Caffe fails even when the batch size is 16. For other GPU platforms, PyTorch and TensorFlow can run with the batch size 64. For Titan Xp, PyTorch obtains 36.5%, 41.7% and 43.4% speedup than TensorFlow when batch sizes are 16, 32 and 64, respectively.

Across different GPU platforms for the VGG-16, Inception-v3, ResNet-50 and DenseNet-121 models, Titan Xp obtains the best performance, which is slightly better than GTX 1080Ti, Titan X (Pascal) and Titan X(Maxwell). Other four GPUs are at least 2 times faster than Titan Z. As shown in Table 1, architectures of Titan Z, Titan X(Maxwell) are Kepler, Maxwell, and the architecture of Titan X(Pascal), GTX 1080Ti and Titan Xp is Pascal. This shows that the more recent GPU architecture is, the faster the running performance is. Although Titan X(Pascal), GTX 1080Ti and Titan Xp have the same GPU architecture, their specifications are more or less different. GTX 1080Ti has better boost clock, memory speed and memory bandwidth than Titan X(Pascal), and Titan Xp has more CUDA cores, higher memory speed, wider memory interface width and memory bandwidth than GTX 1080Ti. This explains why Titan Xp is a little faster than GTX 1080Ti, and GTX 1080Ti is faster than Titan X(Pascal).

Across different frameworks, there are mainly two factors which may result in different computation time. First, in training CNNs, convolutional layers which are the most time-consuming layers are usually invoked by the high performance library cuDNN [46]. However, there are several types of convolution implementations, such as GEMM, FFT and Winograd. TensorFlow prefers to use the Winograd algorithm, and PyTorch could autotune to find the most efficient convolutional algorithm. While Caffe uses GEMM-based convolution. The FFT- and Winograd-based convolutions are faster than GEMM-based convolution in general [24]. The sub-optimal convolution makes Caffe slower than TensorFlow and PyTorch in most cases. Second, PyTorch uses NCHW layout naively which is implicitly supported in cuDNN, while TensorFlow uses NHWC layout. As shown in [48], changing data layout in TensorFlow from NHWC from NCHW leads to 15% speedup. This explains why PyTorch is faster. It is worth noting that PyTorch is a dynamic framework to maximize flexibility, and TensorFlow is a static framework with less computation cost. However, PyTorch minimizes the computational cost of graph construction in every iteration and implements faster GPU kernels for frequent workloads, allowing for more efficient dynamic computation.

4.3.2. Scalability of different deep learning frameworks on multi-GPUs

Since the AlexNet model [11] won the ILSVRC 2012 challenge [35], CNNs have become ubiquitous in various computer vision tasks [49–58]. Making deeper and more complicated networks has been the general trend to improve the performance [12,14,15,34]. However, a single GPU only has a limited memory space, which seriously limits the network capacity. On the other side, training CNNs is time-consuming, especially for deeper networks and large training datasets. Therefore, scalability is very critical for deep learning frameworks, allowing for accelerating the training process by spliting the data across multiple GPUs or machines, and training extremely large models which would not fit into the limited memory of one GPU. Support of multiple GPUs and machines becomes necessary for frameworks. The distributed synchronous stochastic gradient descent (SGD) method [59,60] is widely used to achieve a better scaling performance on multiple GPUs or machines.

In this Section, running time of the VGG-16, Inception-v3, ResNet-50 and DenseNet-121 model is evaluated on 1, 2 and 4 Titan X(Pascal) GPUs. A proper batch size for each model is determined to make the model run on every framework and better utilize the GPU resources. We use the metric (i.e. # of samples processed per second) to measure the throughput. To show the scalability, the speedup is calculated to indicate how much the throughput with GPUs could be increased:

$$speedup = \frac{throughput_{2\times(\#\text{of gpus})} - throughput_{\#\text{of gpus}}}{throughput_{\#\text{of gpus}}} \quad (6)$$

Ideally, the value of speedup should be 100% when the GPU number is doubled.

Results of the VGG-16 model with a batch size 64 per GPU are shown in the top left of Fig. 3. PyTorch has the highest throughput, which can process 127, 203 and 377 images per second as the GPU number increases from 1, 2 to 4, compared with 79, 156 and 316 images in Caffe. TensorFlow has the lowest processing speed on the VGG-16 model, which can process 76, 117 and 196 images per second, respectively. It can be seen that on a single GPU, PyTorch is the best, and Caffe is slightly better than TensorFlow. With the number of GPUs doubls, scalability of Caffe is the most remarkable (97%), while PyTorch and TensorFlow have similar speedup performances, over 50%. When the GPU number is 4, speedup of Caffe is up to 103% compared with 68% in TensorFlow and 86% in PyTorch.

The performance of the Inception-v3 model with a batch size 25 per GPU is shown in the top right of Fig. 3. As for the throughput across deep learning frameworks, PyTorch (114, 192 and 357 images per second on 1, 2 and 4 GPUs) is faster than Caffe (74, 146 and 294 images) and TensorFlow (106, 160 and 279 images). On a single GPU, PyTorch displays a better performance than TensorFlow and Caffe. When the GPU number doubles, speedup of Caffe, TensorFlow and PyTorch is 97%, 51% and 68%. When the number of GPUs increases to 4, Caffe, TensorFlow and PyTorch have 101%, 68% and 86% improvement than the performance on 2 GPUs.

The performance of the ResNet-50 model with a batch size 25 per GPU is shown in the bottom left of Fig. 3. PyTorch can process 464, 268 and 171 images per second on 4, 2 and 1 GPUs, compared with 300, 149 and 77 images in Caffe and 252, 222 and 160 images in TensorFlow. On a single GPU, PyTorch is the fastest compared

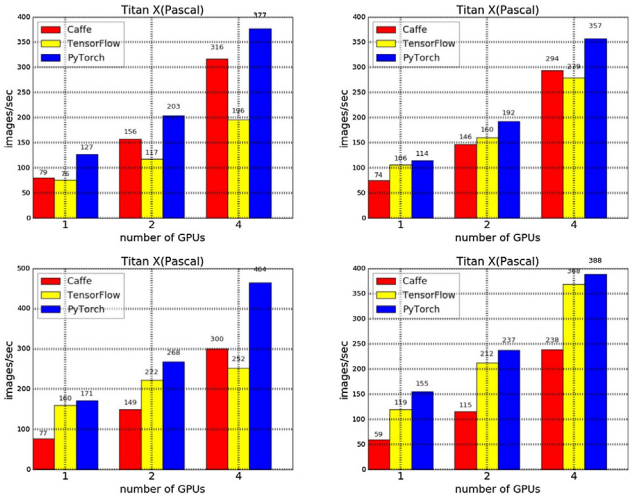

Fig. 3. Scalability comparison of the VGG-16 model with a batch size 64 (top left), the Inception-v3 model with a batch size 25 (top right), the ResNet-50 model with a batch size 25 (bottom left) and the DenseNet-121 model with a batch size 20 (bottom right) w.r.t. multiple GPUs.

with TensorFlow and Caffe. When the number of GPUs doubles, speedup of Caffe, TensorFlow and PyTorch is 94%, 39% and 57%. If the number of GPUs changes to 4, Caffe has the best speedup (101%), while PyTorch is much better than TensorFlow (73% compared with 13.5%).

The batch size per GPU of the DenseNet-121 model is set to 20 in our experiments, and results are shown in the bottom right of Fig. 3. TensorFlow (368, 212 and 119 images per second) has the second-best throughput behind PyTorch (388, 237 and 155), with Caffe lagging relatively far behind (238, 115 and 59). The throughput of PyTorch is slightly better than TensorFlow, and PyTorch and TensorFlow are almost two times faster than Caffe on a single GPU. When the number of GPUs is doubled, scalability of Caffe (95%) is the best, followed by TensorFlow (78%) and PyTorch (53%). With 4 GPUs, Caffe, TensorFlow and PyTorch further achieve 107%, 74% and 64% speedup, respectively.

As for the efficiency of frameworks, TensorFlow implicitly uses NHWC tensor layout. On the contrary, PyTorch adopts NCHW layout naively in which cuDNN is better optimized. This can explain why PyTorch is able to process more images per second than TensorFlow. Meanwhile, PyTorch tends to use FFT-based convolutions which are faster than GEMM-based convolutions used in Caffe [48]. Because the convolutional operation is the computation bottleneck in CNN models, PyTorch is faster than Caffe. Due to its excellent throughput in above experiments, PyTorch is selected to conduct the remaining experiments.

But as for the scalability of different frameworks, it should be noted that the speedup of Caffe is around 100%. We attribute this to the reason that loading images from hard disks instead of more efficient LMDB data format becomes a bottleneck in Caffe. To train CNN models with SGD, the model is iteratively updated with the feeding data. There are four steps generally in one iteration. (1) Load a mini-batch of data from hard disks to CPU memory and transfer the data to GPU memory. The time in this step is represented by t_{data}. (2) Each GPU launches kernels to finish the forward- and backward- operations. (3) The gradients from all GPUs are aggregated. t_{agg} refers to the time of the gradient aggregation. (4) The model in each GPU is updated based on the aggregated gradients. Let t_{gpu} represent the total time in step (2) and (3). It is obvious to represent the time t_{iter} in one iteration with the following equation:

$$t_{iter} = t_{data} + t_{gpu} + t_{agg} \tag{7}$$

Since loading images for the next iteration can be overlapped with GPU computation for the current iteration, and gradient aggregation on the previous iteration can be parallelized with backward operation on the current iteration. So the average iteration time in Caffe is:

$$t_{iter} = max\{t_{data}, t_{gpu}, t_{agg}\} \tag{8}$$

Because we receive info "waiting for data" during training in Caffe, prefetching data is too slow for the next iteration, and GPU computation and gradient aggregation needs to wait for the data loading in every iteration. In other words, t_{data} is larger than t_{gpu} and t_{agg}. When more GPUs are used, GPU computation and gradient aggregation are still hidden behind the data loading, which leads to a bottleneck in the scaling performance. Therefore a linear speedup is achieved in Caffe.

Efficient data communication between GPUs is important to ease the overhead of gradient synchronization: TensorFlow and PyTorch use the Googles Remote Procedure Call (gRPC) [61] Library and NVIDIA Collective Communications Library (NCCL) [62], respectively. Since NCCL has a higher efficiency and lower latency in collective gradient communications [63], TensorFlow has a suboptimal scaling performance compared with PyTorch.

4.4. Comparison of different training datasets

In this section, we investigate the effect of different training datasets (i.e. UMDFaces, WebFace, VGGFace2 and MS-Celeb-1M) on five test datasets (i.e. LFW, VGGFace2-test, IJB-A quality, DFW and UMDFaces). Experimental results are shown in Tables 14 and 15.

Based on comparative results of different frameworks, PyTorch is selected to run experiments in this section because of its high throughput and relatively good accuracies. Due to its good performance, the ResNet-50 is used as the baseline model. The model is trained with 14 epochs. The learning rate is set to 0.1 and divided by 10 every 6 epochs. The batch size is 210 on 3 GPUs. In order to compare training datasets fairly, overlapped subjects between training and test datasets are removed. After removing, an overview of MS-Celeb-1M, UMDFaces, VGGFace2, and WebFace datasets is presented in Table 16. In terms of breadth (number of subjects), MS-Celeb-1M dataset (84,936) gets the first place, followed by WebFace dataset (10,182), VGGFace2 dataset (8410), and then UMDFaces dataset (5222). As for depth (number of images per subject), VGGFace2 dataset (374) is significantly superior than both UMDFaces (45), MS-Celeb-1M (45) and WebFace (39) datasets. The depth of UMDFaces, MS-Celeb-1M and WebFace datasets are very similar.

For the LFW verification task, MS-Celeb-1M dataset achieves the best performance, 0.1% improvement compared with the VGGFace2 dataset. The performance of WebFace dataset slightly lags behind, while UMDFaces dataset ranks the last by a large margin. This indicates that MS-Celeb-1M, VGGFace2 and WebFace datasets are more suitable for this task. This may be because these three datasets are broader compared to UMDFace dataset, allowing the model to learn more discriminative features to increase interclass distances and verify if one pair of images belong to the same subject or not. It is worth noting that the LFW dataset was an early benchmark for face verification. Its performance is almost saturated [21,6]because most faces in LFW dataset are close to frontal. MS-Celeb-1M and VGGFace2 datasets may have good potentials for more challenging tasks.

For the cross-pose face matching task on VGGFace2-pose and UMDFaces-test, the VGGFace2 dataset outperforms the UMDFaces for training by a significant margin. WebFace and MS-Celeb-1M datasets lag behind. Since modern deep learning is heavily data-driven, the generalization performance depends on the distribution of the training dataset [64]. It is known that the data generation pipeline in VGGFace2 dataset aims to encourage age and pose diversity for each subject [23]. Pose information is provided in UMDFaces dataset which has more pose variations compared to the WebFace [26]. Although both UMDFaces and VGGFace2 datasets have pose information, their large gaps on depth and breadth explains why VGGFace2 dataset achieves much better performances than UMDFaces dataset for cross-pose matching tasks. In contrast, despite a large number of images in the MS-Celeb-1M dataset, most of them are frontal faces because of without specific data collection rules in the data collection stage.

For the cross-quality face matching task on IJB-A, VGGFace2 dataset ranks the first on the Low2High task, and MS-Celeb-1M gets the first place on the Middle2High task, respectively, followed by WebFace dataset and then UMDFaces dataset. Because images in these four datasets are crawled from the Internet, they contain various quality levels. If there are more images in the training dataset, there are more images with different image qualities, which guide the model to learn robust features to various image qualities. This explains why VGGFace2 and MS-Celeb-1M datasets are substantially superior to WebFace and UMDFaces. On the other side, MS-Celeb-1M dataset (cleaned by DeepGlint) tends to have high ratios about middle- and high- level image qualities than

Table 14
Experimental results of the ResNet-50 trained on the UMDFaces, WebFace and VGGFace2 datasets and tested on the LFW, VGGFace2-test and IJB-A quality datasets.

Dataset	LFW (%)	VGGFace2 (Similarity Score)						IJB-A (%), different FARs					
		Pose				Age		Low2High			Middle2High		
			Front	Three-quarter	Profile		Young	Mature	1%	0.1%	0.01%	1%	0.1%	0.01%
UMDFaces	95.5	Front	0.783	0.755	0.555	Young	0.707	0.436	40.8	19.5	7.7	69.4	45.5	29.3
		Three-quarter	0.760	0.793	0.645	Mature	0.452	0.743						
		Profile	0.558	0.642	0.749									
WebFace	98.2		0.783	0.761	0.601		0.707	0.438	54.5	29.8	6.9	82.3	62.2	41.8
			0.767	0.793	0.677		0.464	0.762						
			0.611	0.683	0.739									
VGGFace2	98.8		**0.820**	**0.792**	**0.673**		**0.748**	**0.497**	**81.8**	**62.8**	**38.1**	94.8	85.4	69.0
			0.798	**0.826**	**0.736**		**0.522**	**0.790**						
			0.678	**0.740**	**0.786**									
MS-Celeb-1M	**98.9**		0.776	0.734	0.610		0.683	0.412	76.2	59.5	37.6	**95.0**	**89.1**	**77.2**
			0.739	0.777	0.684		0.431	0.739						
			0.615	0.685	0.741									

Table 15
Experimental results of the ResNet-50 trained on the UMDFaces, WebFace and VGGFace2 datasets and tested on the DFW and UMDFaces-test datasets.

Dataset	DFW (%), different FARs		UMDFaces (%), different FARs								
			Easy			Moderate			Difficult		
	1%	0.1%	1%	0.1%	0.01%	1%	0.1%	0.01%	1%	0.1%	0.01%
UMDFaces	21.4	7.6	76.5	59.3	44.9	65.6	43.0	23.8	56.7	32.6	19.4
WebFace	**50.6**	**26.6**	69.8	54.3	42.7	57.7	36.8	22.6	48.0	24.4	10.4
VGGFace2	49.0	25.3	**80.3**	**66.6**	**54.5**	**72.2**	**55.9**	**43.5**	**66.0**	**43.4**	**24.7**
MS-Celeb-1M	26.8	8.4	66.9	49.4	37.1	55.1	30.2	14.5	47.1	22.9	8.3

Table 16
An overview of MS-Celeb-1M, UMDFaces, VGGFace2, and WebFace datasets, after removing overlapped subjects with test datasets.

Dataset	# of subjects	# of images	# of images per subject (average)
MS-Celeb-1M	84,936	3,838,654	45
UMDFaces	5222	236,856	45
VGGFace2	8410	3,141,890	374
WebFace	10,182	393,700	39

VGGFace2 dataset, resulting in a better performance in the Middle2High task.

For the DFW challenge, WebFace dataset performs better than other two datasets, although it is just marginally better than VGGFace2 dataset. The reason may be that images from WebFace dataset are not manually identity labeled and appropriate noise level makes the model more robust to recognize the disguised faces and reject impersonators. VGGFace2 dataset contains much more information and has closer gap with DFW dataset than UMDFaces dataset in terms of depth and breadth, resulting in the remarkable performance improvement. On the other side, MS-Celeb-1M dataset cleaned by DeepGlint (http://trillionpairs.deepglint.com/) may have fewer disguised faces in the dataset, leading to an inferior performance.

For the cross-age face matching task on VGGFace2-age dataset, VGGFace2 dataset obviously obtains a better performance, achieving around 3% improvement because it contains more age variations within the same subject during the image crawling stage [23]. As shown in Table 14, for these three training datasets, the performance improvement is more significant when face matching across young and mature faces, which is more difficult than young to young and mature to mature face matching. Besides, young to young matching has more remarkable improvement compared with mature to mature matching. These results can be attributed to the reason that more young faces in VGGFace2 dataset enable the model to learn more age-invariant features, especially for young to mature face matching. Although WebFace dataset has more images than UMDFaces dataset, these images are more likely to deem as mature faces. This explains why mature to mature face matching has better accuracy improvement, followed by young to mature matching. Young to young face matching performance remains almost the same for WebFace and UMDFaces datasets. The ratio of mature faces in the MS-Celeb-1M dataset may be much higher than young faces. Therefore, MS-Celeb-1M dataset performs poorly in the cross-age task.

Because VGGFace2 dataset takes depth and breadth into consideration, guaranteeing rich intra-subject variations and inter-subject diversity. Moreover, it collects face images with a wide range of poses and ages. This explains why it obtains the best accuracy on cross-pose, cross-age and cross-quality face matching tasks. Since a large number of high-quality faces in the revised MS-Celeb-1M dataset, it achieves the best result on the LFW verification task. For the DFW challenge, some outliers in WebFace dataset make CNN more robust to disguised faces and impersonators, so WebFace provides slightly better performance than VGGFace2. Although UMDFaces dataset has a smaller number of images than WebFace dataset, it has more pose variations and performs better on the cross-pose face matching task.

5. Conclusions and future work

We have performed a benchmark of various models (i.e. AlexNet-v1, AlexNet-v2, VGG-16, VGG-16-BN, GoogLeNet, Inception-v3, ResNet-50, DenseNet-121 and LightCNN-29, Center-loss, SphereFace, CosFace and ArcFace) for face recognition under different frameworks (i.e. PyTorch, TensorFlow and Caffe) in two folds. First, we compare the accuracy of different CNN models within the same deep learning framework. Experimental results show that models, such as the Center-loss, SphereFace, CosFace, ArcFace, GoogLeNet, Inception-v3 and ResNet-50 models achieve

better performances. Second, we compare the accuracy of the same CNN model under different frameworks. Results show that the PyTorch based models tend to obtain the best performance among these three frameworks because of its better weight initialization methods and data pre-processing steps, followed by TensorFlow and then Caffe.

The running time performance of frameworks PyTorch, TensorFlow and Caffe, and GPU platforms are compared as well. PyTorch has the highest throughput and TensorFlow obtains slightly better throughput than Caffe in most cases because of the difference of data layout and convolution implementations among these frameworks. For five GPU platforms, five different designs (e.g. architecture, CUDA core number, memory speed and bandwidth) are tested and we found that the Titan Xp displays slightly faster speed than GTX 1080Ti, Titan X(Pascal) and Titan X(Maxwell) and Titan Z is nearly 2 times slower than the other four GPUs. We also compare the scalability of different frameworks. Experiments show that Caffe has almost linear speedup because of the data loading bottleneck. Because the communication library used in PyTorch (i.e. NCCL) is better than TensorFlow (i.e. gRPC), PyTorch has a slightly better scaling performance.

Four training datasets (i.e. UMDFaces, WebFace, VGGFace2 and MS-Celeb-1M) are investigated for training the CNN models, and five test datasets (i.e. LFW, VGGFace2-test, IJB-A, DFW and UMDFaces-test) are used for recognition accuracy measures. Because of its rich intra-subject variations and inter-class diversity, age and pose information, VGGFace2 dataset is preferable to train CNN models; MS-Celeb-1M dataset (http://trillionpairs.deepglint.com/) has more high-quality frontal faces, which makes it suitable for the controlled face verification task, like access control; Due to the larger pose variations, UMDFaces dataset achieves a good performance on pose related tasks than the WebFace dataset; WebFace allows CNNs to learn more robust features, distinguishing disguised faces and impersonators and surpassing UMDFaces and VGGFace2 datasets on the DFW challenge.

Further, a set of deep face models trained on the WebFace dataset under three frameworks will be made publicly available. Users can take them as baselines to further fine-tune them for their tasks. This will save the training time significantly required for face recognition models, and helpful for beginners.

In future, we plan to evaluate the performance of distributed frameworks over multi-machine environments. And also, newer state-of-the-art face recognition models will be added continuously with more and more newer GPU platforms.

Declaration of Competing Interest

None.

Acknowledgment

The work was partly supported by a NSF CITeR grant.

References

[1] T. Ahonen, A. Hadid, M. Pietikainen, Face description with local binary patterns: application to face recognition, TPAMI (12) (2006) 2037–2041.
[2] J. Lu, V.E. Liong, J. Zhou, Simultaneous local binary feature learning and encoding for homogeneous and heterogeneous face recognition, TPAMI 40 (8) (2017) 1979–1993.
[3] Y. Duan, J. Lu, J. Feng, J. Zhou, Context-aware local binary feature learning for face recognition, TPAMI 40 (5) (2018) 1139–1153.
[4] M. Yang, Q. Wang, W. Wen, Z. Lai, Multi-feature joint dictionary learning for face recognition, in: ACPR, IEEE, 2017, pp. 629–633.
[5] G.B. Huang et al., Labeled faces in the wild: A database for studying face recognition in unconstrained environments, Tech. rep., Technical Report 07–49, University of Massachusetts, Amherst, 2007.
[6] J. Deng, J. Guo, N. Xue, S. Zafeiriou, Arcface: additive angular margin loss for deep face recognition, in: CVPR, 2019, pp. 4690–4699.
[7] J. Yangqing et al., Caffe: Convolutional architecture for fast feature embedding, in: Proceedings of the 22nd ACM International Conference on Multimedia, ACM, 2014, pp. 675–678.
[8] A. Martín et al., Tensorflow: a system for large-scale machine learning, in: OSDI, vol. 16, 2016, pp. 265–283.
[9] P. Adam et al., Automatic differentiation in pytorch, in: NIPS-W, 2017.
[10] F. Chollet, Monthly arxiv.org mentions for frameworks, 2018. URL https://twitter.com/fchollet/status/951828914103402497.
[11] K. Alex et al., Imagenet classification with deep convolutional neural networks, in: NIPS, 2012, pp. 1097–1105.
[12] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, arXiv preprint arXiv: 1409.1556.
[13] S. Christian et al., Going deeper with convolutions, in: Cvpr, 2015.
[14] H. Kaiming et al., Deep residual learning for image recognition, in: CVPR, 2016, pp. 770–778.
[15] H. Gao et al., Densely connected convolutional networks, in: CVPR, vol. 1, 2017, p. 3.
[16] W. Xiang et al., A light cnn for deep face representation with noisy labels, arXiv preprint arXiv: 1511.02683.
[17] W. Qiangchang et al., Learning channel inter-dependencies at multiple scales on dense networks for face recognition, arXiv preprint arXiv: 1711.10103.
[18] J. Lu, J. Hu, Y.-P. Tan, Discriminative deep metric learning for face and kinship verification, TIP 26 (9) (2017) 4269–4282.
[19] W. Yandong et al., A discriminative feature learning approach for deep face recognition, in: ECCV, Springer, 2016, pp. 499–515.
[20] L. Weiyang et al., Sphereface: deep hypersphere embedding for face recognition, in: CVPR, vol. 1, 2017.
[21] H. Wang et al., Cosface: large margin cosine loss for deep face recognition, arXiv preprint arXiv: 1801.09414.
[22] Y. Duan, J. Lu, J. Zhou, Uniformface: learning deep equidistributed representation for face recognition, in: CVPR, 2019, pp. 3415–3424.
[23] C. Qiong et al., Vggface2: a dataset for recognising faces across pose and age, arXiv preprint arXiv: 1710.08092.
[24] B.F. Klare et al., Pushing the frontiers of unconstrained face detection and recognition: Iarpa janus benchmark a, in: CVPR, 2015, pp. 1931–1939.
[25] D.T. Indulal et al., Recognizing disguised faces: human and machine evaluation, PloS One 9 (7) (2014) e99212.
[26] B. Ankan et al., Umdfaces: an annotated face dataset for training deep networks, in: IJCB, IEEE, 2017, pp. 464–473.
[27] H.-W. Ng, S. Winkler, A data-driven approach to cleaning large face datasets, in: ICIP, IEEE, 2014, pp. 343–347.
[28] Y. Dong et al., Learning face representation from scratch, arXiv preprint arXiv: 1411.7923.
[29] O.M. Parkhi et al., Deep face recognition, in: BMVC, vol. 1, 2015, p. 6.
[30] K.-S. Ira et al., The megaface benchmark: 1 million faces for recognition at scale, in: CVPR, 2016, pp. 4873–4882.
[31] G. Yandong et al., Ms-celeb-1m: a dataset and benchmark for large-scale face recognition, in: ECCV, Springer, 2016, pp. 87–102.
[32] S. Shaohuai et al., Benchmarking state-of-the-art deep learning software tools, in: Cloud Computing and Big Data (CCBD), 2016 7th International Conference on, IEEE, 2016, pp. 99–104.
[33] A. Krizhevsky, One weird trick for parallelizing convolutional neural networks, arXiv preprint arXiv: 1404.5997.
[34] S. Christian et al., Rethinking the inception architecture for computer vision, in: CVPR, 2016, pp. 2818–2826.
[35] R. Olga et al., Imagenet large scale visual recognition challenge, IJCV 115 (3) (2015) 211–252.
[36] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, arXiv preprint arXiv: 1502.03167.
[37] L. Jingtuo et al., Targeting ultimate accuracy: face recognition via deep embedding, arXiv preprint arXiv: 1506.07310.
[38] G.B. Huang, E. Learned-Miller, Labeled Faces in the Wild: Updates and New Reporting Procedures, Dept. Comput. Sci., Univ. Massachusetts Amherst, Amherst, MA, USA, Tech. Rep, 2014, 14-003.
[39] C. Jiansheng et al., Face image quality assessment based on learning to rank, IEEE Signal Process. Lett. 22 (1) (2015) 90–94.
[40] G. Guo, N. Zhang, What is the challenge for deep learning in unconstrained face recognition?, in: Automatic Face & Gesture Recognition (FG 2018), 2018 13th IEEE International Conference on, IEEE, 2018, pp. 436–442.
[41] Z. Kaipeng et al., Joint face detection and alignment using multitask cascaded convolutional networks, IEEE Signal Process. Lett. 23 (10) (2016) 1499–1503.
[42] D. Sandberg, Facenet, 2019. https://github.com/davidsandberg/facenet.
[43] F. Claudio et al., Investigating nuisances in dcnn-based face recognition, TIP 27 (11) (2018) 5638–5651.
[44] Y. Wu, K. He, Group normalization, arXiv preprint arXiv: 1803.08494.
[45] H. Kaiming et al., Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, in: ICCV, 2015, pp. 1026–1034.
[46] C. Sharan et al., cudnn: efficient primitives for deep learning, arXiv preprint arXiv: 1410.0759.
[47] N. Silberman, S. Guadarrama, Tensorflowslim image classification model library, 2017.
[48] K. Heehoon et al., Performance analysis of cnn frameworks for gpus, Performance Analysis of Systems and Software (ISPASS).
[49] H. Kaiming et al., Mask r-cnn, in: ICCV, IEEE, 2017, pp. 2980–2988.
[50] Y. Ding et al., Depth-aware saliency detection using convolutional neural networks, J. Vis. Commun. Image Represent. 61 (2019) 1–9.

[51] P. He et al., Frame-wise detection of relocated i-frames in double compressed h. 264 videos based on convolutional neural network, J. Vis. Commun. Image Represent. 48 (2017) 149–158.
[52] T. Wu, S. Tang, R. Zhang, Y. Zhang, Cgnet: A light-weight context guided network for semantic segmentation, arXiv preprint arXiv: 1811.08201.
[53] W. Qiangchang et al., Multiscale rotation-invariant convolutional neural networks for lung texture classification, IEEE J. Biomed. Health Informat. 22 (1) (2018) 184–195.
[54] T. Wu, S. Tang, R. Zhang, J. Cao, J. Li, Tree-structured kronecker convolutional network for semantic segmentation, in: ICME, IEEE, 2019, pp. 940–945.
[55] J. Wu, H. Zheng, B. Zhao, Y. Li, B. Yan, R. Liang, W. Wang, S. Zhou, G. Lin, Y. Fu, et al., Large-scale datasets for going deeper in image understanding, in: 2019 IEEE International Conference on Multimedia and Expo (ICME), IEEE, 2019, pp. 1480–1485.
[56] T. Wu, S. Tang, R. Zhang, G. Guo, Y. Zhang, Consensus feature network for scene parsing, arXiv preprint arXiv: 1907.12411.
[57] M. Jiang, G. Guo, Body weight analysis from human body images, TIFS 14 (10) (2019) 2676–2688, https://doi.org/10.1109/TIFS.2019.2904840.
[58] C. Zalluhoglu, N. Ikizler-Cinbis, Region based multi-stream convolutional neural networks for collective activity recognition, J. Vis. Commun. Image Represent. 60 (2019) 170–179.
[59] C. Jianmin et al., Revisiting distributed synchronous sgd, arXiv preprint arXiv: 1604.00981.
[60] D. Dipankar et al., Distributed deep learning using synchronous stochastic gradient descent, arXiv preprint arXiv: 1602.06709.
[61] Google, Google's remote procedure call library (grpc). http://www.grpc.io.
[62] NVIDIA, Nccl, 2017. https://github.com/NVIDIA/nccl.
[63] A.A. Ahmad et al., Scalable distributed dnn training using tensorflow and cuda-aware mpi: characterization, designs, and performance evaluation, arXiv preprint arXiv: 1810.11112.
[64] H. Chen et al., Learning deep representation for imbalanced classification, in: CVPR, 2016, pp. 5375–5384.