Adaptive Inference for Face Recognition leveraging Deep Metric Learning-enabled Early Exits

Nikolaos Passalis
Dept. of Informatics
Aristotle University of Thessaloniki, Thessaloniki, Greece
passalis@csd.auth.gr

Anastasios Tefas
Dept. of Informatics
Aristotle University of Thessaloniki, Thessaloniki, Greece
tefas@csd.auth.gr

***Abstract*—Deep Learning (DL) models that support adaptive computational graphs allow for easily adapting the computations to the available resources by selecting the most appropriate computational path. However, such models are typically used in classification settings, e.g., using early exits, despite that DL models often aim at extracting representations, e.g., for face recognition. In this work, we provide a metric-learning oriented early exit methodology for DL models. As we demonstrate, employing early exits in metric learning scenarios pose unique challenges compared to existing methodologies for classification-oriented early exits. To this end, we employ the Bag-of-Features model to efficiently extract compact representations from any layer of a DL model that is then combined with an efficient linear regressor to match the final representation of the model (without having to feedforward the whole computational graph). The proposed method is agile and can be directly used with any pre-trained DL model, while it is end-to-end differentiable, allowing for further fine-tuning the models towards having multiple early exits. The effectiveness of the proposed method is demonstrated using five face verification/recognition datasets.**

***Index Terms*—Adaptive Inference, Early Exits, Deep Metric Learning, Lightweight Deep Learning**

I. Introduction

A number of impressive applications, ranging from accurate robotics perception [1] to precise disease prognosis and diagnosis [2], are enabled by powerful Deep Learning (DL) models [3]. Indeed, DL models are becoming increasingly powerful, following the continuous improvements in dedicated hardware accelerators, such as Graphics Processing Units (GPUs) and Tensor Processing Units (TPUs) [4], which allowed for training and deploying deeper and more complicated models. However, in many applications, such as robotics [1] and Internet-of-Things (IoT) [5], we are often still limited to using less powerful hardware, due to a number of limitations, ranging from energy and power constraints to constrained physical form factors. As a result, numerous methods have been proposed to allow for developing more *lightweight* DL models that will be deployed in such devices, while meeting critical application-specific requirements, such as low latency and real-time operation.

These methods include quantization [6], for reducing the number of bits spent for each of the parameters of the model, pruning methods [7], that discard parts of the model that are not critical for its operation, models that are lightweight by design [8], [9], as well as knowledge distillation approaches [10], [11], which aim to transfer the knolwedge from a larger and more complex neural network into a smaller and faster one. These approaches led to more lightweight models that could operate faster in many embedded and mobile devices. However, most of these methods are not capable of adapting to varying computational loads. In other words, the inference time is constant regardless the environmental conditions, e.g., the difficulty of each sample, the load of the system, etc.

This is a critical limiting factor in a number of embedded applications, where the load dynamically varies according to the environmental conditions. For example, for a face recognition application the time needed for face recognition depends on the number of faces that appear in a given frame. As a result, even through a model might operate in real time for a specific number of faces, e.g., 2-3 faces, this might not be the case when a larger number of people appears in a given frame. Therefore, in such cases, we need models that can effectively adapt to the current conditions, providing faster (and possibly less accurate) predictions when the load is higher in order to satisfy the processing time limitations of a given application. In this way, the models can provide accurate answers, exploiting all the available processing time, while still meeting the requirements of each application when the load is higher.

These limitations can be addressed by using models that support *adaptive computational graphs*, such as [12]–[14]. These approaches work by altering the number of computations in order to keep the load within certain limits. This is usually achieved by using multiple paths over the computation graph of the model. Among the most straightforward ways to achieve this is by using *early exits* [12]–[14]. By placing such early *classification* layers at various intermediate layers of the network we can early stop the computation whenever it is deemed appropriate (e.g., when the computational budget is spent or when the network is already confident enough regarding the provided prediction), obtaining an estimation for the representation that would be extracted from the final output layer of the network.

Even though early exits provided a very powerful tool to address the aforementioned limitations, its use is currently limited to classification settings [12]–[14]. However, in many cases, deep learning models aim at extracting representations (metric learning) [15], instead of directly predicting the class to which the input sample belongs to. Perhaps the most well known example of such metric learning task is face recognition [16] and content-based information retrieval [17]. To the best of our knowledge there has been no attempt to use early exits in such scenarios, such as metric learning-based face verification. Even though it could be argued that using early exits in such scenarios could be deemed redundant, since we can directly extract a representation from any layer of a DL model without any modification, we demonstrate that this naive approach has significant limitations and that we can achieve higher accuracy by employing appropriately designed and trained early exit layers.

The main contribution of this work is to provide a metric learning-oriented early exit methodology for DL models. As we experimentally demonstrate, employing early exits in metric learning scenarios pose unique challenges compared to existing methodologies for classification-oriented early exits. To this end, in this work we leverage the Bag-of-Features model to efficiently extract compact representations from any layer of a neural network. Then, an additional small linear regressor is used to regress the final output of the model at selected points of its computational graph. In this way, a representation that can be used in place of the final representation can be readily extracted from an early exit. This provides significant advantages over existing metric learning approaches, which would require keeping a separate database for the representations extracted from each exit layer, increasing the space required for using any additional layer as an early exit and reducing the accuracy of the resulting models, as we demonstrate in Section III. The proposed method is agile and can be directly used with any pre-trained metric learning DL model, while it is end-to-end differentiable, allowing for further fine-tuning the models towards having multiple early exits. The effectiveness of the proposed method is demonstrated using five face verification/recognition datasets, including DL models trained on the large-scale MS-Celeb-1M dataset [18] and evaluated using a wide range of datasets, as well as experiments conducted on two embedded platforms typically used in robotics applications.

The rest of the paper is structured as follows. The proposed method is introduced in Section II. Then, Section III provides the experimental setup and experimental evaluation, while Section IV concludes this paper.

II. Proposed Method

In this Section we present the proposed method. First, we introduce the used notation and then we present the proposed early exit strategy. Let $f_{\mathbf{W}}(\mathbf{x}, i)$ denote the response of the i-th layer of a neural network that is composed of a total of m layers. We used the notation $\mathbf{W}$ to refer to the trainable parameters of the network, while $\mathbf{x}$ denotes the input to the network. In this work we focus on convolutional neural networks that handle images as input, i.e., $\mathbf{x} \in \mathbb{R}^{W \times H \times C}$, where W is the width, H is the height, and C is the number of channels of the image. However, this is without loss of generality, since the proposed method can be directly employed for any other type of DL model. To simplify the used notation, we denote by $\mathbf{y}^{(i)} = f_{\mathbf{W}}(\mathbf{x}, i) \in \mathbb{R}^{W_i \times H_i \times C_i}$ the output of the i-th layer, where W_i, H_i and C_i refer to the width, height and number of channels of the extracted feature map. The notation $\mathbf{y} = f_{\mathbf{W}}(\mathbf{x}, m) \in \mathbb{R}^M$ is used to refer the final output of the network, where M is the dimensionality of the output layer. Finally, we use the notation $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$ to refer to a training set of N images that will be used for training the early exits.

For the rest of this Section, we assume that the network has already been trained to perform a specific metric learning task [16], [19], and we will focus on training the early exits on top of the representations $\mathbf{y}^{(i)}$ extracted at specific points of its computational graph. Early exits typically employ an additional estimator, fitted on top of the representation extracted at various points of the computation graph of the model, to predict the final output of the model. Therefore, we can use an estimator $g^{(i)}_{\mathbf{W}_i}(\cdot)$ at the i-th layer of the network as:

$$g^{(i)}_{\mathbf{W}_i}\left(\mathbf{y}^{(i)}\right) = g^{(i)}_{\mathbf{W}_i}(f_{\mathbf{W}}(\mathbf{x}, i)) \in \mathbb{R}^M. \quad (1)$$

The notation $\mathbf{W}_i$ is used to refer to the parameters each early exit. Classification-based early exits are trained to directly solve the original classification task of the network [12]–[14]. However, for metric-learning oriented network this approach cannot be employed, since even if we use the original loss used for training the network, e.g., the contrastive loss [19], we will not learn representations in the same space as the one formed by the last layer of the network. As a result, the representations extracted by the early exits would not be useful for performing queries in a database that consists of representations extracted from the final layer of the network.

To overcome this limitation, in this work we proposed using a distillation inspired approach [10], i.e., to train the early exits in order to mimic the output of each layer. Perhaps the most straightforward approach to ensure that the features extracted by the early exits $g^{(i)}_{\mathbf{W}_i}(\cdot)$ will reside in the same space as the final representation of the network $\mathbf{y}$ is to minimize the quadratic divergence between these two representations. Therefore, early exit estimators are trained in order to minimize the following loss:

$$\mathcal{L}_i = \frac{1}{N} \sum_{j=1}^{N} ||\mathbf{y}_j - g^{(i)}_{\mathbf{W}_i}(\mathbf{y}^{(i)}_j)||_2^2, \quad (2)$$

where i denotes the early exit that we are training, $|| \cdot ||_2$ denotes the l_2 norm of a vector and the notation $\mathbf{y}_j$ is used to refer to the representation extracted when the j-th sample is fed into the network, i.e., $\mathbf{y}_j = f_{\mathbf{W}}(\mathbf{x}_j, m) \in \mathbb{R}^{N_C}$.

Usually, early exits employ a feature aggregation approach to reduce the dimensionality of the extracted feature maps, e.g., Global Average Pooling, that is then followed by a fully connected layer. However, naive feature aggregation approaches, such as global average/max pooling, have been shown to discard useful information [20]. Therefore, in this work we employ a Bag-of-Features (BoF)-based aggregation layer in order to reduce the dimensionality of the extracted feature maps and extract a compact summary representation that can be further adapted towards the task at hand [21].

BoF-based pooling works as follows. First, we quantize each feature vector extracted from a feature map using a set of N_K codewords. In this work, we use the notation $\mathbf{v}_{ij}$ to denote each codeword, where i is the layer to which the codeword belongs to (a separate codebook is used for each layer) and j refers to a specific codeword (out of N_K possible ones). In this way, we can then extract a membership vector for each feature vector that belongs to the i-th exit as:

$$[\mathbf{u}_{ikl}]_j = \frac{K([\mathbf{y}^{(i)}]_{kl}, \mathbf{v}_{ij})}{\sum_{m=1}^{N_K} K([\mathbf{y}^{(i)}]_{kl}, \mathbf{v}_{im})} \in [0, 1]. \tag{3}$$

The notation (k, l) is used to refer to the location of the feature map from which the feature vector is extracted, while $K(\cdot)$ denotes the kernel used for measuring the similarity between codewords and feature vectors. In this work, we use a Gaussian-based kernel to this end:

$$K(\mathbf{x}, \mathbf{v}_{ij}) = \exp(-\frac{||\mathbf{x} - \mathbf{v}_{ij}||^2}{2\sigma_i^2}), \tag{4}$$

where σ_i is a trainable scaling factor that scales the distances between feature vectors and codewords to the appropriate range. For non-trainable BoF models, σ_i is typically set to the average distance between the feature vectors and the codewords.

After extracting the membership vectors $\mathbf{u}_{ikl}$ we can directly extract a compact histogram representation for each exit as:

$$\mathbf{s}^{(i)} = \frac{1}{W_i H_i} \sum_{k=1}^{W_i} \sum_{l=1}^{H_i} \mathbf{u}_{ikl} \in \mathbb{R}^{N_K}, \tag{5}$$

This histogram representation provides a summary of the concepts that appear in the corresponding features. By appropriately tuning the codewords we can *focus* the representation on different concepts. For example, using *k-means* to learn the codebook leads to a generic representation that can be used for any task, while finetuning the whole layer using gradient descent allows for learning task-specific codewords (provided that the BoF layer is part of a network trained for a specific task). Finally, this histogram representation is fed into a linear layer that projects the histogram into the desired space, i.e., $g^{(i)}_{\mathbf{W}_i}(\mathbf{y}^{(i)}) = s^{(i)}(\mathbf{y}^{(i)})\mathbf{W}^l_i$, where $\mathbf{W}^l_i \in \mathbb{R}^{N_K \times M}$. In the case of metric-learning networks this would be the space formed by the output layer of the network. Then, early exits can be trivially trained using gradient descent, minimizing the loss provided in (2).

III. Experimental Evaluation

The proposed method was evaluated using the MS-Celeb-1M [18], Labeled Faces in the Wild (LFW) [22], [23], Cross Pose LFW (CPLFW) [24], Cross Age LFW (CALFW) [25], and VGGFace2 [26] datasets. More specifically, we follow a standard face verification setup [27], where the models are trained on the MS-Celeb-1M dataset and evaluated on the remaining four datasets, i.e., LFW, CPLFW, CALFW and VGGFace2. All images used for the conduncted experiments were resized to 112×112 pixels. For the evaluation procedure we randomly sample 6,000 image pairs that either correspond to face images of the same person or to face images of different persons (equally distributed among the two cases). A face pair is considered to belong to the same person when the distance between the corresponding embeddings is lower than a certain threshold. This threshold is selected to maximize the face verification accuracy on a validation dataset. As a result, we report the average 10-fold cross validation accuracy for all the conducted experiments, i.e., the threshold is selected according to the validation split and the accuracy is reported on the corresponding test set. For all the conducted experiments we used a ResNet-50 network [28], where inverted residual blocks were employed for improving its efficiency [29]. The early exits were placed after the 1st, 2nd and 3nd residual block. The dimensionality of the feature vectors extracted from each of these blocks is 128, 256 and 256 respectively, while the dimensionality of the final representation of the network is 512.

TABLE I
Number of additional parameters required for each method

Method	**Exit 1**	**Exit 2**	**Exit 3**
Raw*	1.2-1.4M	2.4-2.7M	2.4-2.7M
LR	66k	131k	131k
BoF	327k	393k	393k
Proposed	327k	393k	393k

*For the raw method we assume that we employ a database that contains 3,000 feature vectors, which corresponds to the evaluation setup used in this paper.

Three different methods were evaluated along with the proposed one. For the first one we directly extracted the feature vectors from each early exit, we employed global average pooling and then we queried the database using these representations. This approach is called "*raw*" in the rest of this paper, since it relies on directly using the raw feature vectors, as they are extracted from the network. Even though the dimensionality of these feature vectors is lower, this method requires keeping a separate database with the feature vectors extracted from each additional early exit, significantly increasing the storage requirements as shown in Table I. Next, we evaluated a linear regressor (denoted by "LR") that was trained to directly regress the output representation of the network based on the (average) pooled representation extracted from each early exit. The same approach was also repeated using the Bag-of-Feature model, where we used 512 codewords for building the codebook (using the k-means algorithms and the feature vectors extracted from each early exit for building the codebook). This method is denoted as "BoF". Note that both the LR and BoF methods can be regarded as a simplified (ablated) version of the proposed one, since, to the best of our knowledge, neither has been proposed in the literature for constructing early exits. Despite this, they consist a strong baseline, as we demonstrate later in this Section.

Finally, we evaluated the proposed method using again $N_K = 512$ codewords (in order to be directly comparable with the BoF baseline). The number of training iterations was set to 2,000 with a learning rate of 0.001 for the parameters of the BoF model, while the linear regressor was fine-tuned using a learning rate of 0.0001. The Adam algorithm with its default parameters was used for the optimization [30], while the batch size was set to 32. After training the BoF-based layers, the linear regressor was fitted again using the closed form solution to ensure a fair comparison with the LR method (16,000 training samples were used for fitting the regressor). A comparison between the number of parameters required for adding the early exits to the base model are summarized in Table I. All the methods (LR, BoF, Proposed) significantly reduce the number of required parameters over the naive raw baseline. Using the BoF layer increases, to a small extent, the number of required parameters, but as we demonstrate later, this is also accompanied by a corresponding increase in the face verification accuracy. Furthermore, in all cases (except from the raw baseline) the number of parameters is kept within reasonable limits. It is also worth noting that for the BoF/Proposed method the number of the added parameters can be further controlled by decreasing the number of codewords N_K.

The experimental evaluation is provided in Table II. The four evaluated methods are compared on four different datasets using the three different added early exits. In all the cases, using a subsequent early exit increases the obtained verification accuracy as expected. Furthermore, just using a linear regressor (LR) to regress the final representation of the network leads to a significant increase over directly using the raw representation. Indeed, in some cases (e.g., CALFW) the accuracy increases by over 25%. Then, using the BoF model further increases the performance, while employing the proposed method leads to the overall best accuracy in all the evaluated cases. It is worth noting that in some cases, the verification performance is very close to the actual performance of the final output of the network, as reported in Table III.

To further verify that using the proposed method leads to actual performance improvements we evaluated all the employed methods using two embedded platforms, the NVIDIA Jetson TX-2 and the NVIDIA Jetson AGX. The obtained results are summarized in Table IV. Indeed, using early exits leads to a significant speedup , e.g., about 4× for the first exit compared to the final output of the network. Using the proposed method leads to a slight overhead (about 10%) compared to the raw and LR methods. However, it still leads to enormous performance improvements over the final output of the network, while achieving higher verification accuracy, as demonstrated before.

TABLE II
FACE VERIFICATION ACCURACY ON FOUR DIFFERENT DATASETS USING THREE DIFFERENT EARLY EXITS. THE MEAN AND STANDARD DEVIATION OF THE 10-FOLD CROSS-VALIDATION ACCURACY IS REPORTED.

Method	Exit 1	Exit 2	Exit 3
Dataset: LFW			
Raw	68.60 ± 2.25	82.33 ± 1.25	92.80 ± 1.57
LR	75.05 ± 2.00	88.98 ± 1.42	94.15 ± 0.69
BoF	79.93 ± 1.59	92.33 ± 1.22	92.58 ± 1.23
Proposed	$\mathbf{80.98 \pm 2.64}$	$\mathbf{92.70 \pm 1.28}$	$\mathbf{96.58 \pm 0.64}$
Dataset: CPLFW			
Raw	52.75 ± 1.95	51.98 ± 1.74	63.83 ± 2.38
LR	66.80 ± 1.71	75.57 ± 2.04	83.02 ± 1.36
BoF	68.52 ± 1.79	79.98 ± 2.16	81.13 ± 1.05
Proposed	$\mathbf{68.98 \pm 1.18}$	$\mathbf{80.05 \pm 1.70}$	$\mathbf{84.20 \pm 1.56}$
Dataset: CALFW			
Raw	55.00 ± 1.62	61.72 ± 1.50	68.10 ± 1.98
LR	70.28 ± 1.51	82.18 ± 1.89	87.35 ± 1.18
BoF	71.80 ± 1.18	84.55 ± 1.06	84.93 ± 1.51
Proposed	$\mathbf{73.65 \pm 1.82}$	$\mathbf{84.78 \pm 1.50}$	$\mathbf{89.37 \pm 1.37}$
Dataset: VGGFace2			
Raw	56.88 ± 2.07	64.24 ± 1.12	78.88 ± 1.43
LR	67.10 ± 1.74	78.50 ± 1.85	84.16 ± 1.33
BoF	68.86 ± 1.84	81.56 ± 2.69	83.50 ± 1.55
Proposed	$\mathbf{71.34 \pm 2.04}$	$\mathbf{82.32 \pm 2.32}$	$\mathbf{88.10 \pm 1.43}$

TABLE III
FACE VERIFICATION ACCURACY ON FOUR DIFFERENT DATASETS USING THE FINAL OUTPUT OF THE EMPLOYED NETWORK.

Dataset	Accuracy
LFW	99.78 ± 0.22
CPLFW	92.05 ± 1.36
CALFW	95.78 ± 1.20
VGGFace2	94.98 ± 0.71

The mean and standard deviation of the 10-fold cross-validation accuracy is reported.

IV. CONCLUSIONS

In this work we proposed a metric learning-oriented early exit methodology that can be effectively used with any DL model. To this end, we employed the Bag-of-Features model in order to extract compact representations from enormous feature maps, that were then fed into lightweight linear regressors trained to approximate the final representaiton of the model. In this way, it is possible to estimate the final output of a large and complex DL model at various points of its computational graph. As we experimentally demonstrated, this approach allows for effectively adapting the computations to the available resources. At the same time, the proposed method works significantly better than naive approaches that were used until now, such as directly using the representation extracted from a layer and building separate databases for each layer. Furthermore, the proposed method is easy to use and agile, since it can be readily combined with any DL model. At the same time, it can be used to train the models in an end-to-end fashion, in order to better adapt to the task at hand, since it

TABLE IV
SPEED (FPS) COMPARISON BETWEEN DIFFERENT METHODS AND EXITS

Method	Exit 1	Exit 2	Exit 3
Jetson TX-2 (≈ 0.8 FP32 TFLOPS)			
Default		5.6	
Raw	23.4	13.2	9.6
LR	23.1	13.2	9.6
BoF/Proposed	20.8	12.9	9.5
Jetson AGX (≈ 1.4 FP32 TFLOPS)			
Default		8.6	
Raw	41.2	24.5	18.7
LR	40.8	24.3	18.7
BoF/Proposed	38.1	23.9	18.4

Frames Per Second (FPS) are reported. FPS are measured using batches of 4 input images, to ensure more consistent measurements. The average of 100 runs is reported.

relies on a fully differentiable formulation.

In this way, the proposed method paves the way for building more advanced early exit methodologies for representation learning tasks. For example, the early exits at subsequent layers can be trained to regress the residual error [20], instead of the raw final representation. We expect that this will allow for further increasing the performance of the method, since each exit could be finetuned to just refine the previous output. Furthermore, the models can be also used to adapt the model to the difficulty of each sample, in a similar fashion as in [31], allowing for skipping layers of the networks for easier samples. This could allow for further increasing the inference speed and reducing the energy requirements of the models.

Acknowledgments: This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871449 (OpenDR). This publication reflects the authors' views only. The European Commission is not responsible for any use that may be made of the information it contains.

REFERENCES

[1] N. Sünderhauf, O. Brock, W. Scheirer, R. Hadsell, D. Fox, J. Leitner, B. Upcroft, P. Abbeel, W. Burgard, M. Milford *et al.*, "The limits and potentials of deep learning for robotics," *The Intl. Journal of Robotics Research*, vol. 37, no. 4-5, pp. 405–420, 2018.

[2] B. Gecer, S. Aksoy, E. Mercan, L. G. Shapiro, D. L. Weaver, and J. G. Elmore, "Detection and classification of cancer in whole slide breast histopathology images using deep convolutional networks," *Pattern Recognition*, vol. 84, pp. 345–356, 2018.

[3] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015.

[4] Y. E. Wang, G.-Y. Wei, and D. Brooks, "Benchmarking tpu, gpu, and cpu platforms for deep learning," *arXiv preprint arXiv:1907.10701*, 2019.

[5] M. Mohammadi, A. Al-Fuqaha, S. Sorour, and M. Guizani, "Deep learning for iot big data and streaming analytics: A survey," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 2923–2960, 2018.

[6] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," in *Proc. Intl. Conf. on Learning Representations*, 2016.

[7] J.-H. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," in *Proc. IEEE Intl. Conf. on Computer Vision*, 2017, pp. 5058–5066.

[8] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[9] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 2018, pp. 6848–6856.

[10] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *NIPS Deep Learning and Representation Learning Workshop*, 2015.

[11] N. Passalis and A. Tefas, "Learning deep representations with probabilistic knowledge transfer," in *Proc. European Conf. on Computer Vision*, 2018, pp. 268–284.

[12] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," in *Proc. Intl. Conf. on Pattern Recognition*, 2016, pp. 2464–2469.

[13] N. Passalis, J. Raitoharju, A. Tefas, and M. Gabbouj, "Adaptive inference using hierarchical convolutional bag-of-features for low-power embedded platforms," in *Proc. IEEE Intl. Conf. on Image Processing*, 2019, pp. 3048–3052.

[14] Y. Bai, S. S. Bhattacharyya, A. P. Happonen, and H. Huttunen, "Elastic neural networks: A scalable framework for embedded computer vision," in *Proc. European Signal Processing Conf.*, 2018, pp. 1472–1476.

[15] K. Sohn, "Improved deep metric learning with multi-class n-pair loss objective," in *Proc. Intl. Conf. on Neural Information Processing Systems*, 2016, pp. 1857–1865.

[16] H. V. Nguyen and L. Bai, "Cosine similarity metric learning for face verification," in *Proc. Asian Conf. on Computer Vision*, 2010, pp. 709–720.

[17] A. Krizhevsky and G. E. Hinton, "Using very deep autoencoders for content-based image retrieval." in *Proc. European Symposium on Artificial Neural Networks*, vol. 1, 2011, p. 2.

[18] "MS-Celeb-1M."

[19] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *Proc. Intl. Conf. on Machine Learning*, 2020, pp. 1597–1607.

[20] N. Passalis, J. Raitoharju, A. Tefas, and M. Gabbouj, "Efficient adaptive inference for deep convolutional neural networks using hierarchical early exits," *Pattern Recognition*, vol. 105, p. 107346, 2020.

[21] N. Passalis and A. Tefas, "Learning bag-of-features pooling for deep convolutional neural networks," in *Proc. IEEE Intl. Conf. on Computer Vision*, 2017, pp. 5766–5774.

[22] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller, "Labeled faces in the wild: A database for studying face recognition in unconstrained environments," University of Massachusetts, Amherst, Tech. Rep. 07-49, October 2007.

[23] G. B. Huang, V. Jain, and E. Learned-Miller, "Unsupervised joint alignment of complex images," in *Proc. Intl. Conf. on Computer Vision*, 2007.

[24] T. Zheng and W. Deng, "Cross-pose lfw: A database for studying cross-pose face recognition in unconstrained environments," *Beijing University of Posts and Telecommunications, Tech. Rep*, vol. 5, 2018.

[25] T. Zheng, W. Deng, and J. Hu, "Cross-age lfw: A database for studying cross-age face recognition in unconstrained environments," *arXiv preprint arXiv:1708.08197*, 2017.

[26] Q. Cao, L. Shen, W. Xie, O. M. Parkhi, and A. Zisserman, "VGGFace2: A dataset for recognising faces across pose and age," in *Proc. IEEE Intl. Conf. on Automatic Face & Gesture Recognition*, 2018, pp. 67–74.

[27] Face.evoLVe.PyTorch. [Online]. Available: https://github.com/ZhaoJ9014/face.evoLVe.PyTorch

[28] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[29] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 2018, pp. 4510–4520.

[30] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[31] N. Passalis, J. Raitoharju, M. Gabbouj, and A. Tefas, "Efficient adaptive inference leveraging bag-of-features-based early exits," in *Proc. Intl. Workshop on Multimedia Signal Processing*, 2020, pp. 1–6.