ESANN 2022 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium) and online event, 5-7 October 2022, i6doc.com publ., ISBN 978287587084-1. Available from http://www.i6doc.com/en/.

Feature Compression Using Dynamic Switches in Multi-split CNNs

Suresh Kirthi Kumaraswamy[1], Alexey Ozerov[1,2], Ngoc Q. K. Duong[1,3], Anne Lambert[1], François Schnitzler[1] and Patrick Fontaine[1] *

[1] InterDigital, Inc. - Rennes, France,
[2] Ava - Paris, France, [3] Lacroix Impulse - Rennes, France.
The present work has been done while all authors were with InterDigital, Inc.

Abstract. Convolutional neural networks (CNN) are often computationally demanding for mobile devices. Offloading some computation lowers this burden: initial convolutional layers are processed on a smartphone, the resulting high dimensional features are transmitted, and latter layers are processed in the cloud/edge/another device. To improve this process, we propose *Dynamic Switch*, a convolutional subnetwork enabling *anywhere splittable* CNNs with *multirate* feature compression using a *single* set of network parameters. We achieve 90% feature compression with at most 3% accuracy loss for MobileNet and MSDNet on ImageNet dataset and at most 4.58% on CIFAR100 dataset with MSDNet, ResNet-18, MobileNet/MobileNetv2 and ShuffleNet/ShuffleNetv2.

1 Introduction

Convolutional Neural Networks (CNN) are increasingly deployed in constrained environments such as mobile devices and Internet of Things (IoT) devices with limited memory and computational resources and/or transmission bandwidth. One possible solution to facilitate such deployments is to split the computation between devices, edge, cloud etc. [1]. A CNN can be split such that one part of the processing takes place in the constrained device and the feature vector thus generated is transmitted for further processing in the next device or the cloud/edge server [2]. However, the bandwidth consumed in such scenarios might be important since the feature size at an intermediate layer can be higher than the input data dimension [3]. It is therefore important to reduce the size of the intermediate features for the purpose of efficient transmission to the next device/edge/cloud. This transmission problem is generally solved by adding a compression module for the features [4, 5, 6]. However, most approaches consider a fixed location and/or a fixed compression rate, thus at inference time the compressed features are not well-adapted to the changing communication bandwidth [5] or require a different model for each deployment setting (characteristics of each device and of the transmission network).

In this work, we present a multiple splits solution for CNN architectures by adding a compression/expansion module at each split point and call this module Dynamic Switch (DySw). Inspired by the Slimmable approach [7, 8],

*This work has been supported by European Union's Horizon 2020 research and innovation program under grant number 951911 - AI4Media.

ESANN 2022 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium) and online event, 5-7 October 2022, i6doc.com publ., ISBN 978287587084-1. Available from http://www.i6doc.com/en/.

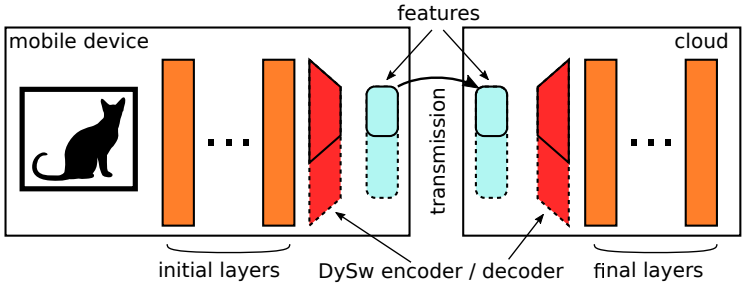

Fig. 1: Illustration of the use of DySw with $N = 2$ compression rates. Depending on the constraints, either some (full shapes) or all (full + dashed shapes) compressed features are transmitted from the mobile device to the cloud.

it can realize different compression and expansion ratios using a single set of parameters. These ratios are switchable at inference time so that the compressed features can be adapted to the available transmission bandwidth. But unlike [7], our purpose is not to slim the whole model but rather to compress the CNN features. Also, a CNN may use several DySw units at difference split locations and they can be added to a pretrained network by training only them.

Two works [9, 10] are close to our proposal. In [9], Huang *et al.* propose feature dimension reduction technique, referred as progressive slicing. This concept is similar to ours wherein a method to adapt feature dimension is considered. At a pre-specified split point, a progressive slice layer is pre-configured for different feature dimensions. The main difference is that separate network parameters per slice configuration are maintained unlike us where we maintain a single shared set. In [10], Shao *et al.* introduce an autoencoder of six layers, three each for encoder and decoder. Here the dimension of the features is adjusted depending on the communication channel conditions and noise level. They choose the dimensions with highest magnitudes by thresholding the chosen dimensions' changes for each input. Unlike them, in our method retained dimensions are fixed and pre-learnt. In addition, our autoencoder consists of just two hidden layers, one per encoder and decoder. This results in a lightweight and more flexible approach.

Our main contributions are: (1) We propose a multi-split CNN for distributed inference, each with multi-rate feature compression. (2) We investigate the use of different losses and strategies for the parameter estimation. (3) We validate the benefit of the DySw within several popular CNN architectures on classification task. (4) Finally, we empirically show that the DySw can be augmented to a pretrained CNN without significant loss in accuracy.

2 Proposed Approach

Assuming here, without loss of generality, an image classification task, let's consider a CNN $f : x \to y$, where $x \in \mathbb{R}^{m \times n \times 3}$, $y \in \mathbb{R}^c$ and function f maps input

ESANN 2022 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium) and online event, 5-7 October 2022, i6doc.com publ., ISBN 978287587084-1. Available from http://www.i6doc.com/en/.

image x to output y. x is an input image with m rows, n columns, 3 channels; y is a c-dimensional vector representing c classes, and f is parameterized by θ. Let the DySw be $g_{(l-1)} : x_l \rightarrow x'_l$, where $g_{(l-1)}$ is inserted at the end of the $(l-1)^{th}$ layer of the CNN f and x'_l is the output of DySw $g_{(l-1)}$. The DySw $g_{(l-1)}$ is made up of a convolutional (conv) layer with kernel size 1×1 that projects the input on to a lower dimension and another conv layer of kernel size 1×1 which expands and re-instates the compressed feature to the dimension of the original input $x_{(l)}$. A typical DySw subnetwork implementation is made up of one conv layer, one ReLU layer and one batchnorm layer for the compression and a conv, ReLU and a batchnorm layer for expansion.

The DySw is capable of switching among different compression expansion rates through width selection in the conv layers. This is illustrated in Figure 1 with only 2 compression rates for clarity. The blocks in the figure refer generally to conv layers or multiple conv layers with associated non linearity and normalization layers. The DySw operates in N compression rates indexed by $K_i \in \{K_1, \ldots, K_N\}$ with K_1 being least compression and K_N highest compression. In each compression rate K_i, only the first K_i kernels are used. Hence the size of the compressed vector is K_i.

We considered various training losses. We tested the cross-entropy (CE) loss alone or in combination with classical Knowledge Distillation (KD) [11] loss where the least/uncompressed rate output supervises the outputs due to other compression rates, as in slimmable approaches [7]. Since the DySw are structurally similar to autoencoders, we also considered mean square error (MSE) as a reconstruction loss. When the network is trained from scratch the cross-entropy loss gives the best results but when DySw is added to a pretrained network a combination of CE, KD and MSE works best.

3 Experimental Results

We implement DySw in six representative classes of architectures. A DySw is placed at the end of: blocks in MSDNet [12], *residual-blocks* in ResNet-18 [13], *depthwise convolutions* in MobileNet [14], *Linear Bottlenecks* in the MobileNetv2 [15], and stages in ShuffleNet and ShuffleNetv2 [16, 17]. ResNet is conventional, MobileNet and ShuffleNet are compact and MSDNet is an anytime inference CNN based on the DenseNet architecture family. Each DySw realises multiple compression ratios (here we consider three: 50%, 75% and 90%).

We experimented with the CIFAR100 dataset [18] having 50K training and 10K test images of size $32 \times 32 \times 3$ with 100 classes and successfully validated our results on ImageNet dataset [19] classification dataset (1000 classes, 1.28M training images and 50K validation images) for a compact model (MobileNet) and a large one (MSDNet).

3.1 Adding Dynamic Switches doesn't significantly impact accuracy

We trained DySw CNNs from scratch, with standard data augmentation and provide Top-1 accuracy (model answer must be expected answer) as is standard

ESANN 2022 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium) and online event, 5-7 October 2022, i6doc.com publ., ISBN 978287587084-1. Available from http://www.i6doc.com/en/.

Compression Rate	**50%**	**75%**	**90%**	**Uncompressed**
MSDNet†	70.05	69.42	69.21	73.79
MSDNet*	65.26	64.82	63.71	66.01
MobileNet†	63.91	64.21	64.59	66.07
MobileNet*	69.56	69.41	69.0	70.2
MobileNetv2†	65.05	64.95	65.09	67.26
ShuffleNet†	69.44	69.52	67.96	69.26
ShuffleNetv2†	70.1	69.99	69.05	70.09
ResNet-18†	71.93	71.77	71.78	75.95

Table 1: Top-1 accuracy at different compression rates for DySw CNNs trained from scratch. † stands for CIFAR-100 and * for ImageNet.

Compression Rate	**50%**	**75%**	**90%**	**Un-C**
CLIO [9] *MobileNetv2*	92	86	NA	NA
Ours *MobileNetv2*	91.54	91.53	91.21	93
LCOT [10] *ResNet18*	NA	91.83	NA	NA
Ours *ResNet18*	93.81	93.74	93.67	94.1

Table 2: Top-1 accuracy at different compression rates compared against CLIO[9] on MobileNetv2 and LCOT[10] on ResNet18.

practice in Image Recognition tasks. Results, listed in Table 1, show that the decrease in accuracy is limited in all configurations. For example, the largest gap (4.58%) is observed for CIFAR-100, with 90% compression rate for the features for MSDNet. Interestingly, the gap is smaller for ImageNet than for CIFAR-100, suggesting our approach scales well to large images and/or data sets.

3.2 DySw is more accurate and flexible than Existing Methods

Here we present a comparison of our method to two prior methods [9, 10] that are similar in spirit to ours. Though our caveat here is that both these methods operate in the regime of single split with multiple rates of compression [9] or single split with single rate of compression [10]. Despite realizing multiple splits and multiple compression rates, our method performs either almost as good in one criterion (of 50% compression) or mostly better in the others.

Huang *et al.* [9] (CLIO) perform experiments mostly on MobileNetv2 with CIFAR-10 and ImageNet-20 (self-curated dataset of 20-random classes). We therefore consider CIFAR-10 results for comparison as the classes chosen for ImageNet are unavailable. We compare our method with [9] (CLIO) in the top three rows of Table- 2. Clearly CLIO performs better than our method at 50% feature dimension compression, but on the higher compression regimes our method performs better. The reason for CLIO's better accuracy could be due to dedicated network parameters at 50% compression. Our performance

ESANN 2022 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium) and online event, 5-7 October 2022, i6doc.com publ., ISBN 978287587084-1. Available from http://www.i6doc.com/en/.

Training Type	From Scratch				From Pretrained		
Compression Rate	**50%**	**75%**	**90%**		**50%**	**75%**	**90%**
DySw@Block-1	72.00	71.89	71.88		71.66	71.45	69.75
DySw@Block-2	71.98	71.87	71.76		71.6	71.9	69.6
DySw@Block-3	72.00	71.80	71.89		71.57	71.69	69.6
DySw@Block-4	72.00	71.74	71.29		71.53	71.52	71.38

Table 3: Accuracy while enabling a single DySw at various locations for ResNet18, either trained from scratch or with pretrained initialization on CIFAR100. DySw@Block-1 means DySw is used after Block-1.

on high compression regimes could be attributed to joint training of different compression rates compared to per compression rate training in CLIO.

Shao *et al.* [10] show the effectiveness of LCOT on ResNet (we assume it is ResNet18) using CIFAR-10 and MNIST. We consider CIFAR-10 for our comparison as it is a more challenging dataset than MNIST and the results are shown in the last two rows of Table 2. Our method outperforms LCOT by 2%.

3.3 Additional experiments

Using pretrained networks and training only DySW sublayers rather than training the full network from scratch is possible. Some results are given in Table 3 for ResNet-18. Results with other architectures are similar.

Learning one autoencoder (1AE) by compression rate decreased accuracy significantly compared to our approach. Out of 21 configurations on MSDNet, 1AE was respectively better/worse in 3/18 configurations by up to 2%/10%. 1AE would also use much more memory to store the different models.

Training a single DySw at one position where the model is split, as opposed to training all DySw at all possible positions, improved accuracy by up to 3%, widening the gap with other approaches. However, this requires one training procedure for every split.

4 Conclusions

One way to realize practical deployment of CNNs in connected devices and smart phones is through collaborative processing using cloud/edge/other devices. This can be achieved by splitting CNNs. But feature sizes at the split points are usually large and require feature compression for efficient transmission over standard communication channels.

In this setting, we presented a module called *Dynamic Switch* (DySw) that can switch among various feature compression rates during inference, enabling adaptation to changing transmission constraints. We trained from scratch CNNs with multiple DySw, hence supporting multiple slit points. Also, we investigated augmenting DySw in pretrained CNNs by training only the DySw. We showed

ESANN 2022 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium) and online event, 5-7 October 2022, i6doc.com publ., ISBN 978287587084-1. Available from http://www.i6doc.com/en/.

that in both cases we almost retain the performance of the CNNs with minimal accuracy loss.

We successfully demonstrated the potential of our methods on many popular CNN architectures like MSDNet, MobileNets, ShuffleNets and ResNet-18. One downside of DySw is the long training time due to the multiple forward passes per training iteration. Thus, investigating more efficient training algorithms would be interesting for future improvement.

References

[1] S. Teerapittayanon, B. McDanel, and H.-T. Kung. Distributed deep neural networks over the cloud, the edge and end devices. In *ICDCS*, 2017.

[2] H. Li, C. Hu, J. Jiang, Z. Wang, Y. Wen, and W. Zhu. Jalad: Joint accuracy-and latency-aware deep structure decoupling for edge-cloud execution. In *ICPADS*, 2018.

[3] I. V. Bajić, W. Lin, and Y. Tian. Collaborative intelligence: Challenges and opportunities. In *ICASSP*, 2021.

[4] A. E. Eshratifar, A. Esmaili, and M. Pedram. BottleNet: A Deep Learning Architecture for Intelligent Mobile Cloud Computing Services. In *ISLPED*, 2019.

[5] Y. Dong, P. Zhao, H. Yu, C. Zhao, and S. Yang. Cdc: Classification driven compression for bandwidth efficient edge-cloud collaborative deep learning. *arXiv:2005.02177*, 2020.

[6] J. Choi, H. J. Chang, T. Fischer, S. Yun, K. Lee, J. Jeong, Y. Demiris, and J. Y. Choi. Context-aware deep feature compression for high-speed visual tracking. In *CVPR*, 2018.

[7] J. Yu, L. Yang, N. Xu, J. Yang, and T. Huang. Slimmable neural networks. *arXiv:1812.08928*, 2018.

[8] J. Yu and T. S. Huang. Universally slimmable networks and improved training techniques. In *ICCV*, 2019.

[9] J. Huang, C. Samplawski, D. Ganesan, B. Marlin, and H. Kwon. Clio: enabling automatic compilation of deep learning pipelines across iot and cloud. In *MobiCom*, 2020.

[10] J. Shao, Y. Mao, and J. Zhang. Learning task-oriented communication for edge inference: An information bottleneck approach. *arXiv:2102.04170*, 2021.

[11] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv:1503.02531*, 2015.

[12] G. Huang, D. Chen, T. Li, F. Wu, L. Van Der Maaten, and K. Q. Weinberger. Multi-scale dense convolutional networks for efficient prediction. *arXiv:1703.09844*, 2017.

[13] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[14] 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:1704.04861*, 2017.

[15] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018.

[16] X. Zhang, X. Zhou, M. Lin, and J. Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *CVPR*, 2018.

[17] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *ECCV*, 2018.

[18] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[19] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.