Structured Sparsity on Embedded Devices

Enzo Tartaglione, Gianmarco Nuzzarello, Andrea Bragagnolo and Marco Grangetto
Università degli Studi di Torino
Torino, Italy
email: enzo.tartaglione@unito.it

I. Current size of neural networks and deployment on mobile devices

Deep Neural Networks (DNNs) can solve extremely challenging tasks thanks to complex stacks of (convolutional) layers with thousands of neurons [1], [2], [3].

We can define the complexity of a neural network as both the number of its learnable parameters and number of operations performed at inference time: architectures such as AlexNet and VGG have a complexity in the order of 60 and 130 million parameters and approximately 1 and 15 GFLOPs respectively. These cmplex neural architectures are challenging to deploy in scenarios where resources such as the memory available for deployment or storage are limited.
In order to cope with neural networks memory requirements, inference time and energy consumption, several (complementary) approaches have been proposed, like changing the network topology and enhancing weight sharing [3], [4], quantization strategies [5] and, very recently, pruning.
Pruning is a popular approach aiming to reduce the deep neural networks' number of required parameters. In particular, all the pruning-based techniques aim at detecting redundant or less relevant parameters to remove them [6], [7], [8]. On this topic, a very large amount of strategies have been proposed [9], [10], [11] from which we can identify two main classes:

- *un-structured* strategies, where parameters are greedily pruned, regardless the final model's efficiency;
- *structured* strategies, where parameters are removed in a structured way, with a beneficial effect to the final computational complexity and memory footprint.

Typically, un-structured approaches are able to remove a larger number of parameters than the structured ones [9] and, for this reason, these are currently very popular. Will they bring advantages on edge devices regarding memory footprint and inference time?
The main purpose of the proposed demo is to show the benefits of the structured pruning over un-structured pruning. In particular, we have designed an Android demo app to measure real-time inference time on some trained and pruned architectures, which will also made available to the demo's attendees.

II. Architectures to test

In our demo we will test some architectures trained on image classification tasks. In particular, we will test differences between:

- full baseline model;
- model with UN-structured pruning;
- model with structured pruning.

Please notice that training a smaller architecture from scratch leads to lower generalization capability. Given the state-of-the-art optimizers and loss functions, training a larger model and then pruning it to reduce its size is currently the most promising approach towards having small, well-generalizing models [12].

III. Our Android demo app

Our demo will be centered to the use of *NNinference*, our customized android app (branched from PyTorch Mobile[1]) aiming at running neural networks directly on the mobile device and to measure efficiency in terms of inference time and memory footprint. The main purpose of the demo is to practically observe differences between neural networks having structured and non-structured sparsity, besides baseline models. Towards this end, no weight/activation quantization strategies are employed, since their use is beyond the main scope of the demo.
We plan to share the following trained and pruned architectures:

- ResNet-32, trained on CIFAR-10;
- VGG-16, trained on CIFAR-10;
- AlexNet, trained on CIFAR-100;
- ResNet-101, trained on ImageNet.

Results on inference results are reported in Table I. Some models are currently being pruned (N/A) and all the results reported with "*" are partial results: the final numbers might further improve the FPS / reduce the inference time. Please also notice that for CIFAR-100 and ImageNet the Top-5 error is reported, while for the other architectures the Top-1 error is the error metrics. The inference time and FPS reported are obtained averaging 1k inferences on a Huawei P20 smartphone, equipped with 4x2.36 GHz Cortex-A73 + 4x1.84 GHz Cortex-A53 processors and 4GB RAM, running Android 8.1 "Oreo". For each of the above-mentioned architectures, we are going to provide three different models:

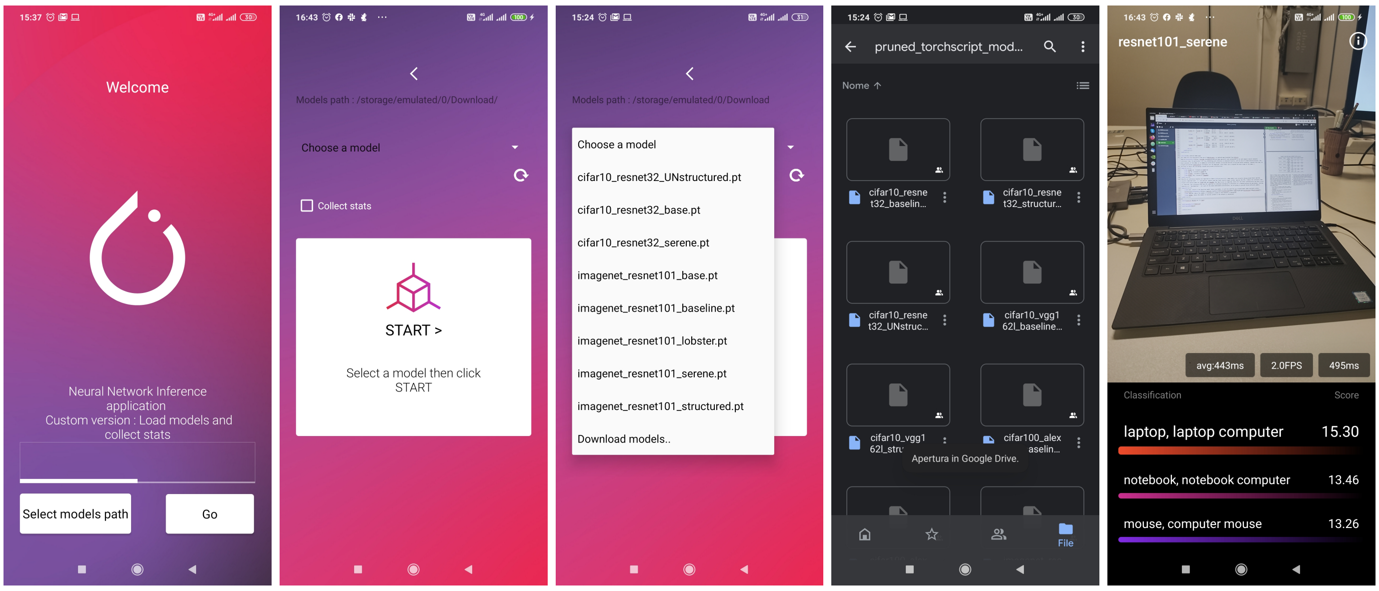
- none, which is the baseline model (where available, it will be used the pre-trained model made available within torchvision[2]);
- UNstructured, where the number of parameters pruned in the network is maximized;
- structured, where the number of neurons pruned in the network is maximized.

An overview describing the app's usage steps is provided in Fig. 1. Along with the demo, we will share a link to directly download the `.apk`: in this way we would grant everyone the possibility to download and test the efficiency of deep models with structured pruning with their own android device. Currently, our app (under development) is open source and available at https://github.com/EIDOSlab/EIDOS-app.

[1]https://github.com/pytorch/android-demo-app.git
[2]https://pytorch.org/docs/stable/torchvision/models.html?highlight=torchvision%20pretrained

TABLE I: Tested architectures. All the models and the app to locally test those will be made available for the demo.

Architecture	Dataset	Pruning	Inference time [ms]	FPS	Memory footprint [MB]	Storage Size [MB]	Error [%]
VGG-16	CIFAR-10	None	204.21	4.9	57.57	51.51	7.36
		UNstructured	N/A				
		Structured	98.67	10.13	11.56	0.97	7.80
ResNet-32	CIFAR-10	None	32.12	31.13	1.84	1.63	7.36
		UNstructured	31.67	31.57	1.83	0.48	7.33
		Structured	24.83	40.27	0.87	0.37	8.09
AlexNet	CIFAR-100	None	131.41	7.61	92.31	79.27	20.09
		UNstructured	N/A				
		Structured	75.27	13.29	43.80	2.47	17.88
ResNet-101	ImageNet	None	957.58	1.04	174.49	156.67	6.44
		UNstructured	956.35	1.05	173.54	35.66	8.24
		Structured	929.77*	1.08*	172.15*	27.84*	9.45*


Welcome
Neural Network Inference application
Custom version : Load models and collect stats
Select models path
Go
Models path : /storage/emulated/0/Download/
Choose a model
Collect stats
START >
Select a model then click START
Choose a model
cifar10_resnet32_UNstructured.pt
cifar10_resnet32_base.pt
cifar10_resnet32_serene.pt
imagenet_resnet101_base.pt
imagenet_resnet101_baseline.pt
imagenet_resnet101_lobster.pt
imagenet_resnet101_serene.pt
imagenet_resnet101_structured.pt
Download models..
pruned_torchscript_mod...
Nome
cifar10_resnet32_baselin...
cifar10_resnet32_structur...
cifar10_resnet32_UNstruc...
cifar10_vgg1 62l_baseline...
cifar10_vgg1 62l_stru
cifar100_alex aselin...
Apertura in Google Drive.
File
resnet101_serene
avg:443ms
2.0FPS
495ms
Classification
Score
laptop, laptop computer 15.30
notebook, notebook computer 13.46
mouse, computer mouse 13.26

(a) (b) (c) (d) (e)

Fig. 1: App usage example. Opening the app (a), it is given as a possibility to set a “Models path”, where all the deep models are or will be stored. After tapping “Go”, you enter the models selection frame (b). It is possible to select a deep network from a drop-down menu (c): all the locally saved models will automatically appear. In such a menu it will also be given the possibility to download more models tapping “Download models..” which redirects to an online shared folder (d). Finally, the model is used at inference time (e), reading the classification results (the bottom displays the top-three results) from the camera input as well as inference time and real-time FPS (mid-right of the screen).

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[2] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 2015.

[3] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[4] Y. Lu, G. Lu, R. Lin, J. Li, and D. Zhang, “Srgc-nets: Sparse repeated group convolutional neural networks,” *IEEE transactions on neural networks and learning systems*, 2019.

[5] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, “Quantized convolutional neural networks for mobile devices,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 4820–4828.

[6] D. P. Kingma, T. Salimans, and M. Welling, “Variational dropout and the local reparameterization trick,” in *Advances in Neural Information Processing Systems*, 2015, pp. 2575–2583.

[7] E. Tartaglione, S. Lepsøy, A. Fiandrotti, and G. Francini, “Learning sparse neural networks via sensitivity-driven regularization,” in *Advances in Neural Information Processing Systems*, 2018, pp. 3878–3888.

[8] S. Han, J. Pool, J. Tran, and W. Dally, “Learning both weights and connections for efficient neural network,” in *Advances in neural information processing systems*, 2015, pp. 1135–1143.

[9] J. Frankle and M. Carbin, “The lottery ticket hypothesis: Finding sparse, trainable neural networks,” *7th International Conference on Learning Representations, ICLR 2019*, 2019.

[10] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, “Rethinking the value of network pruning,” *arXiv preprint arXiv:1810.05270*, 2018.

[11] E. Tartaglione, S. Lepsøy, A. Fiandrotti, and G. Francini, “Learning sparse neural networks via sensitivity-driven regularization,” in *Advances in neural information processing systems*, 2018, pp. 3878–3888.

[12] E. Tartaglione, A. Bragagnolo, and M. Grangetto, “Pruning artificial neural networks: A way to find well-generalizing, high-entropy sharp minima,” in *Artificial Neural Networks and Machine Learning – ICANN 2020*, I. Farkaš, P. Masulli, and S. Wermter, Eds. Cham: Springer International Publishing, 2020, pp. 67–78.