Towards Efficient Capsule Networks

(Article begins on next page)

14 October 2023

TOWARDS EFFICIENT CAPSULE NETWORKS

Riccardo Renzulli   *Marco Grangetto*

Computer Science Department, University of Turin, 10149 Turin, TO, Italy

ABSTRACT

From the moment Neural Networks dominated the scene for image processing, the computational complexity needed to solve the targeted tasks skyrocketed: against such an unsustainable trend, many strategies have been developed, ambitiously targeting performance's preservation. Promoting sparse topologies, for example, allows the deployment of deep neural networks models on embedded, resource-constrained devices. Recently, Capsule Networks were introduced to enhance explainability of a model, where each capsule is an explicit representation of an object or its parts. These models show promising results on toy datasets, but their low scalability prevents deployment on more complex tasks. In this work, we explore sparsity besides capsule representations to improve their computational efficiency by reducing the number of capsules. We show how pruning with Capsule Network achieves high generalization with less memory requirements, computational effort, and inference and training time.

Index Terms— capsule networks, routing, pruning

1. INTRODUCTION

Capsule Networks (CapsNets) [1, 2] were proposed to overcome the shortcomings of Convolutional Neural Networks (CNNs). For instance, CNNs need many training images in different orientation, pose, size etc. to be robust to affine transformations [1]. Moreover, CNNs employ max-pooling to both reduce the dimensions of the input space and to route low-level features deeper in the network. However, the spatial relationships between parts (e.g. the mouth or the nose) of an object (e.g. a human face) within an image is lost because max-pooling progressively drops the spatial information [1]. CapsNets build a hierarchy of parts by means of an iterative routing mechanism [1]. This can be seen as a parallel atten-
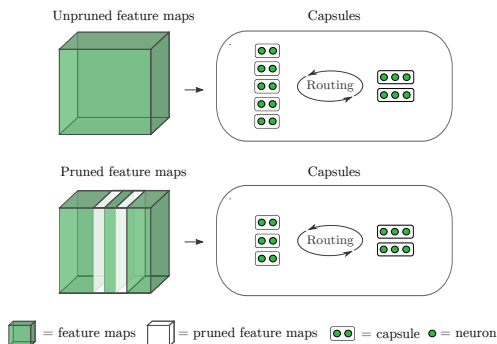
Fig. 1. A standard CapsNet with many capsules (top). We exploit structured pruned representations (bottom) to reduce the number of capsules.

tion mechanism which models the connections between capsules, retaining correct spatial relationships between objects.

Despite CapsNets are successfully applied to various tasks [3, 4], the research is still in its early stages and CapsNets are far from replacing CNNs as state-of-the-art neural neural architectures. In fact, their applicability is limited because stacking multiple capsule layers increase the computational effort [5, 6]. Applying CapsNets on high resolution data with complex backgrounds is also critical: stacking many capsule layers or adding more capsules can lead to instability during training because of the delicate routing algorithm [7]. Finding the connections between many capsules is not trivial and tuning the number of routing iterations during training can be problematic [8]. However, the problem of over-parametrization and the difficulty of deploying networks in scenarios with limited memory and resources is not peculiar only to CapsNets, but also to CNNs. To this end, pruning methods (which reduce the size and complexity of a neural network model) have gained more and more attention lately [9]. Learning sparse topologies can effectively reduce the network footprint and speedup execution with negligible performance loss [10]. So, why not exploiting these techniques also for CapsNets? The reduction their complexity, especially in early layers, will enable their deployment on complex datasets. The aim of this work is to exploit structured pruning approaches to reduce the complexity of CapsNets without performance loss and test them on data with non-straightforward background such as CIFAR10 and high

dimensional data such as Tiny ImageNet. Fig. 1 provides an overall idea of the functioning of our approach. We show how employing structurally sparse backbones, extracting features in early layers for CapsNets, affects the number of capsules in the overall network. Our experiments show that reducing the complexity of the backbone is an effective way to achieve high generalization with less memory requirements, energy consumption, training and inference time.

2. CAPSULE NETWORKS

2.1. Background

CapsNets organize neurons into activity vectors called *capsules*. Each element of these vectors accounts for the instantiation parameters of the object that the capsule represents such as its pose and other properties like thickness, color, deformation, etc. [1]. The probability that a specific object exists in the image is simply the magnitude of its capsule vector [1]. CapsNets are organized in layers and are typically composed by convolutional layers and two or more capsule layers. The CapsNet architecture introduced in [1] is a shallow network with two capsule layers. In this work, we replace both traditional global average pooling layer and final fully connected layer with these two capsule layers. The poses of the first capsule layer (PrimaryCaps) $\boldsymbol{u}_i$, called *primary capsules*, are built upon convolutional layers reshaped into the capsule space. An iterative *routing-by-agreement* mechanism is performed to compute the poses of the capsules of the next layer (OutputCaps). Each capsule $\boldsymbol{u_i}$ makes a prediction $\hat{\boldsymbol{u}}_{j|i}$, thanks to a transformation matrix $\boldsymbol{W}_{ij}$, for the pose of an upper layer output capsule j

$$\hat{\boldsymbol{u}}_{j|i} = \boldsymbol{W}_{ij}\boldsymbol{u}_i. \tag{1}$$

Then, the total input $\boldsymbol{s}_j$ of an output capsule j is computed as the weighted average of votes $\hat{\boldsymbol{u}}_{j|i}$

$$\boldsymbol{s}_j = \sum_i c_{ij}\hat{\boldsymbol{u}}_{j|i}, \tag{2}$$

where c_{ij} is the coupling coefficient between a primary capsule i and an output capsule j. The un-normalized pose $\boldsymbol{s}_j$ of an output capsule j is then "squashed" to compute the normalized pose $\boldsymbol{v}_j$

$$\boldsymbol{v}_j = \text{squash}(\boldsymbol{s}_j) = \frac{\|\boldsymbol{s}_j\|^2}{1 + \|\boldsymbol{s}_j\|^2}\frac{\boldsymbol{s}_j}{\|\boldsymbol{s}_j\|}. \tag{3}$$

The coupling coefficients are determined by a "routing softmax" activation function, whose initial logits b_{ij} are the log prior probabilities the i-th capsule should be coupled to the j-th one. At each iteration, the update rule for the logits is

$$b_{ij} \leftarrow b_{ij} + \boldsymbol{v}_j \cdot \hat{\boldsymbol{u}}_{j|i}. \tag{4}$$

The steps defined in (2), (3), (4) are repeated for r routing iterations. Sabour *et al.* introduced the *margin loss* to train CapsNets [1].

2.2. Related work

Sabour *et al.* [1] proposed a shallow CapsNet to overcome some limitations of CNNs. They achieved state-of-the-art results on toy datasets like MNIST and smallNORB. Recent works focused on building deeper and more efficient architectures of CapsNets in order to apply them on more complex data such as CIFAR10. We can divide these attempts into three groups: i) pure CapsNets, namely deep models where only capsule layers are stacked on top of each other [2, 11, 12]; ii) CapsNets where the primary capsules are extracted with more than one convolutional layer [7, 13]; iii) CapsNets where the primary capsules are extracted with a pretrained state-of-the-art backbone [14, 15].

As regards pure capsule layers, Hinton *et al.* [2] replace the dynamic routing with Expectation-Maximization, adopting matrix capsules instead of vector capsules in order to reduce the number of trainable parameters. They also introduce the idea of convolutional capsules, where the routing is performed locally between a subset of capsules defined by a receptive field. Gugglberger *et al.* [11] introduce residual connections between capsule layers to train deeper capsule networks.

A simple but effective way to build deeper capsule models consists of employing several pure convolutional layers before the last capsule layer. Rajasegaran *et al.* [7] propose a deep CapsNet composed by four layers. All the layers have only one routing iteration (so the coupling coefficients are uniform) except the last one. They approximate the first 3 layers with 2D convolutional layers with skip connections so only one capsule layer exists in their architecture. Mazzia *et al.* [13] extract primary capsules by means of a set of convolutional and Batch Normalization layers, replacing dynamic routing with attention routing. They show that even with a very limited number of parameters their network achieves state-of-the-art results on MNIST, MultiMNIST and smallNORB.

Finally, instead of stacking pure convolutional layers, one can employ state-of-the-arts backbones like ResNets as feature extraction networks. For example, Hahn *et al.* [14] incorporates a *self-routing* method such that capsule models do not require agreements anymore. In all of their experiments, models share the same base CNN architecture, namely a ResNet-20, requiring less computation.

Our work relates to those models where primary capsules are extracted using efficient state-of-the-art backbones like ResNets and MobileNets. We employ structurally pruned backbones to scale CapsNets to bigger datasets and improve their efficiency. We show how these pruned Capsule Networks achieve high generalization with less memory requirements and computational effort.
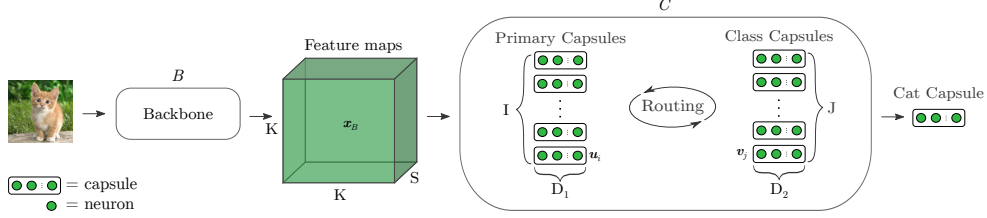
Fig. 2. The architecture used in this work is composed by a backbone part and a capsule part. Primary capsules space has 8 real dimensions while output capsules are 16-dimensional vectors ($D_1 = 8$ and $D_2 = 16$).

3. PROPOSED METHOD

3.1. Primary capsules extraction

Following the work of [14, 15], we extract I primary capsule activity vectors by means of a backbone network. With this approach, we can build deep capsule networks in an efficient way since the input space of the capsule network is significantly reduced. Fig. 2 shows the architecture used in this work. We can build a CapsNet on top of the backbone by replacing the last two layers by a primary capsule (PrimaryCaps) and fully-connected capsule (OutputCaps) layers, respectively. Note that PrimaryCaps is a convolutional capsule layer with I channels of convolutional D_1 capsules, namely each capsule contains D_1 convolutional units with a $K \times K \times S$ kernel. The kernel size used is wide as the spatial dimensions of the output of the backbone network, which means the number of capsules is dynamically reduced. In order to compute output capsules, we use the same routing-by-agreement algorithm described in [1]. In our experiments we use backbones both as fixed feature extractor, namely we freeze their weights except for the capsule layer, and as trainable feature extractor, where we finetune all layers.

3.2. Effect of pruned backbones to capsule layers

In order to reduce the computational complexity of CapsNets as in Fig. 2, one possible strategy is to deploy pruned backbones. Let us define the backbone network $\mathcal{B}(\boldsymbol{x}_\mathcal{M}, \boldsymbol{w}_\mathcal{B})$ where $\boldsymbol{w}_\mathcal{B}$ are the backbone parameters and $\boldsymbol{x}_\mathcal{M}$ is the input; it produces as output the tensor $\boldsymbol{x}_\mathcal{B} \in \mathbb{R}^{K \times K \times S}$. Then, tensor $\boldsymbol{x}_\mathcal{B}$ is fed into the CapsNet $\mathcal{C}(\boldsymbol{x}_\mathcal{B}, \boldsymbol{w}_\mathcal{C})$, where $\boldsymbol{w}_\mathcal{C}$ are the capsule parameters. We can concatenate the two parts expressing the end-to-end model $\mathcal{N}$ as

$$\mathcal{N}(\boldsymbol{x}_\mathcal{M}, \boldsymbol{w}_\mathcal{B}, \boldsymbol{w}_\mathcal{C}) = \mathcal{C}\left[\mathcal{B}(\boldsymbol{x}_\mathcal{M}, \boldsymbol{w}_\mathcal{B}), \boldsymbol{w}_\mathcal{C}\right]. \quad (5)$$

As we saw in Sec. 3.1, D_1-dimensional primary capsules poses $\boldsymbol{u}_i$ are built upon convolutional layers. We define the number of primary capsules as

$$I = \left\lfloor \frac{S}{D_1} \right\rfloor. \quad (6)$$

From this, we observe that reducing the complexity of the backbone would result in the overall reduction of the complexity for the entire model $\mathcal{N}$, and towards this end pruning has already proved to be an effective approach [9, 10].

Pruning approaches can be divided into two groups. *Unstructured* pruning methods aim at minimizing the cardinality $\|\boldsymbol{w}\|_0$ of the parameters in the model, regardless the output topology [16, 17, 18]. On the other hand, *structured* approaches drop groups of weight connections entirely, such as entire channels or filters, imposing a regular pruned topology [9, 19]. As an effect, they minimize the cardinality of some i-th intermediate layer's output $\|\boldsymbol{x}_i\|_0$. Bragagnolo *et al.* [20] showed that structured sparsity, despite removing significantly less parameters from the model, yields lower model's memory footprint and inference time. When pruning a network in a structured way, a simplification step which practically reduces the rank of the matrices is possible; on the other side, encoding unstructured sparse matrices lead to representation overheads [10].

Considering the structure of the primary capsules layer, we are interested in the reduction of the cardinality for the input of the primary capsules $\boldsymbol{x}_\mathcal{B}$. Indeed, as we can see in (6), the minimization of $\|\boldsymbol{x}_\mathcal{B}\|_0$ results in a proportional reduction of S, which in turn reduces I, resulting in a reduction of the votes and the complexity of the routing algorithm [8].
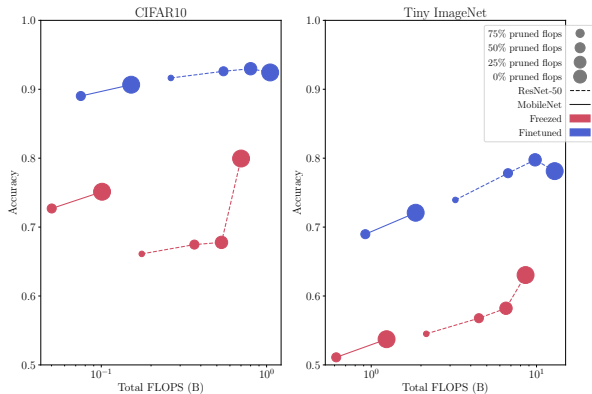
4. EXPERIMENTS

We performed several experiments with different backbones on CIFAR10 and Tiny ImageNet. We choose ResNets and MobileNets networks since they are used as common baselines in traditional computer vision tasks. In fact, they offer a good compromise between both efficiency and performance. Furthermore, they are mainly composed of convolutional layers so it is easy to replace the first convolutional layer in the original CapsNet introduced in [1] with these models.

4.1. Experimental Setup

We resized CIFAR10 images to 64×64 resolution so as to add a deeper backbone network as in [7]. We used the 5%

Backbone pruned FLOPS (%)	Bottleneck size	Primary caps	Total FLOPS (B)	GPU memory consumption (GB)	Training time (epoch, s)	Accuracy
0	2048	256	12.9	47.02	462	0.78
25	1970	246	**9.8**	**43.50**	**450**	**0.80**
50	1947	243	**6.7**	**39.42**	**410**	**0.78**
75	944	118	3.2	20.11	214	0.74

Table 1. Performances using ResNet-50 as backbone (finetuned, TinyImageNet).

Fig. 3. Accuracies using ResNet-50 and MobileNetV1 as backbones for CIFAR10 (left) and Tiny ImageNet (right) with several pruned FLOPS configurations.

of the training set as validation set for hyper-parameters tuning. We also run experiments on Tiny Imagenet (224×224 images, 200 classes). Here we used 10% of the training set as validation set and the original validation set as test set. Our experiments consider several flavors of the architecture in Fig. 2 with different number of primary capsules I, backbones (ResNet-50 e MobileNetV1) and training methods (whether the backbone is finetuned or not). We train all the networks minimizing a *margin loss* [1] with the Adam optimizer and a batch size of 128. The experiments were run on a Nvidia Ampere A40 equipped with 48GB RAM, and the code uses PyTorch 1.10.[1]

We also used PyNVML library to compute the GPU memory consumption. As pretrained pruned backbones, we have used both ResNet-50 and MobileNetV1 pruned with Eagle-Eye [19]. EagleEye is a state-of-the-art, open-source, structured pruning strategy which identifies relevant features at the level of intermediate outputs x_i.

4.2. Results and discussion

The main results are shown in Fig. 3. Each configuration differs from the dataset used, percentage of pruned parameters in the backbone network (which affects the number of primary

---

[1]The source code is available at `https://github.com/EIDOSLAB/towards-efficient-capsule-networks`.

capsules), and training method (backbone finetuned or not). In Fig. 3 we report the accuracy as a function of the total number of FLOPS (of both backbone and capsule part) for training the network. We can see that when the backbone is freezed (in red) the networks achieve poor performances, but as the number of pruned backbone FLOPS decreases, namely the number of primary capsules is higher, the accuracy increases. The poor performance can be explained by the fact that the output features $x_\mathcal{B}$ of the backbone are not directly optimized to represent objects poses. Therefore, they can not be used directly as input to capsule layers. To overcome such a drawback, we can either add more primary capsules or train the whole network. In fact, when the backbone is finetuned (in blue) the performance is higher since the backbone is able to map more suitable features to the capsules space. We can see in this case that adding more capsules does not always lead to an improvement in the classification accuracy. In Tab. 1 the performance for Tiny ImageNet with a pretrained ResNet-50 backbone is reported. Employing a backbone with 50% of pruned FLOPS yields similar accuracy as with a full backbone, with less GPU memory consumption and training time. With 25% of pruned FLOPS we can even improve the accuracy compared to the unpruned one. The partial improving in the performance in a low pruning regime, despite being at a first glance surprising, is not new to the literature and it is twofold. First, Rajasegaran *et al.* [7] shows that building too many capsules inhibites learning because the coupling coefficients are too small, preventing the gradient flow. Furthermore, Han *et al.* observed such a phenomenon for unstructured pruning [16]. Recently, it has been showed that the average entropy in the bottleneck layer for pruned backbones (in our case, $x_\mathcal{B}$) is higher than in non-pruned ones [21]: this results in the propagation of less specific and more general information, which prevents features overfit, on top of which capsules layers can extract much more accurate information.

5. CONCLUSION

In this work we improved the scalability and reduced the computational effort of CapsNets on complex datasets deploying backbones with structured sparsity. CapsNets with many capsules are difficult to train: in such a scenario the routing algorithm, a key mechanism for CapsNets, struggles to find the necessary agreements between capsules. Therefore, the employment of pruned backbones (so fewer capsules) for high

resolution datasets leads to competitive results with no performance loss. We also showed how extracting features in early layers with sparse networks improves efficiency, memory consumption, inference and training time of the overall CapsNet. Therefore, with this work, we open the way to apply CapsNets on resource constrained devices. Future works involve specific methods concerning pruning capsule layers in a structured way.

6. REFERENCES

[1] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton, "Dynamic routing between capsules," in *Advances in Neural Information Processing Systems*. 2017, vol. 30, pp. 3856–3866, Curran Associates, Inc.

[2] Geoffrey E Hinton, Sara Sabour, and Nicholas Frosst, "Matrix capsules with EM routing," in *International Conference on Learning Representations*, 2018.

[3] Yongheng Zhao, Tolga Birdal, Haowen Deng, and Federico Tombari, "3d point capsule networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 1009–1018.

[4] Parnian Afshar, Arash Mohammadi, and Konstantinos N Plataniotis, "Brain tumor type classification via capsule networks," in *2018 25th IEEE International Conference on Image Processing (ICIP)*. IEEE, 2018, pp. 3129–3133.

[5] Chandan Kumar Singh, Vivek Kumar Gangwar, Anima Majumder, Swagat Kumar, Prakash Chanderlal Ambwani, and Rajesh Sinha, "A light-weight deep feature based capsule network," in *2020 International Joint Conference on Neural Networks (IJCNN)*, 2020, pp. 1–8.

[6] Alberto Marchisio, Beatrice Bussolino, Alessio Colucci, Maurizio Martina, Guido Masera, and Muhammad Shafique, "Q-capsnets: A specialized framework for quantizing capsule networks," in *Proceedings of the 57th ACM/EDAC/IEEE Design Automation Conference*. 2020, DAC '20, IEEE Press.

[7] J. Rajasegaran, V. Jayasundara, S. Jayasekara, H. Jayasekara, S. Seneviratne, and R. Rodrigo, "Deepcaps: Going deeper with capsule networks," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 10717–10725.

[8] Riccardo Renzulli, Enzo Tartaglione, Attilio Fiandrotti, and Marco Grangetto, "Capsule networks with routing annealing," in *Artificial Neural Networks and Machine Learning – ICANN 2021*, Cham, 2021, pp. 529–540, Springer International Publishing.

[9] Enzo Tartaglione, Andrea Bragagnolo, Francesco Odierna, Attilio Fiandrotti, and Marco Grangetto, "Serene: Sensitivity-based regularization of neurons for structured sparsity in neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–14, 2021.

[10] Andrea Bragagnolo and Carlo Alberto Barbano, "Simplify: A python library for optimizing pruned neural networks," *SoftwareX*, vol. 17, pp. 100907, 2022.

[11] Josef Gugglberger, David Peer, and Antonio Rodríguez-Sánchez, "Training deep capsule networks with residual connections," in *Artificial Neural Networks and Machine Learning – ICANN 2021*, Cham, 2021, pp. 541–552, Springer International Publishing.

[12] Yi Xiong, Guiping Su, Shiwe Ye, Yuan Sun, and Yi Sun, "Deeper capsule network for complex data," in *2019 International Joint Conference on Neural Networks (IJCNN)*, 2019, pp. 1–8.

[13] Vittorio Mazzia, Francesco Salvetti, and Marcello Chiaberge, "Efficient-capsnet: capsule network with self-attention routing," *Scientific reports*, vol. 11, 2021.

[14] Taeyoung Hahn, Myeongjang Pyeon, and Gunhee Kim, "Self-routing capsule networks," in *Advances in Neural Information Processing Systems*. 2019, vol. 32, pp. 7658–7667, Curran Associates, Inc.

[15] Inyoung Paik, Taeyeong Kwak, and Injung Kim, "Capsule networks need an improved routing algorithm," in *Proceedings of The Eleventh Asian Conference on Machine Learning*, Nagoya, Japan, 17–19 Nov 2019, vol. 101 of *Proceedings of Machine Learning Research*, pp. 489–502, PMLR.

[16] Song Han, Jeff Pool, John Tran, and William Dally, "Learning both weights and connections for efficient neural network," in *Advances in neural information processing systems*, 2015, pp. 1135–1143.

[17] Enzo Tartaglione, Skjalg Lepsøy, Attilio Fiandrotti, and Gianluca Francini, "Learning sparse neural networks via sensitivity-driven regularization," in *Advances in neural information processing systems*, 2018, pp. 3878–3888.

[18] Enzo Tartaglione, Andrea Bragagnolo, Attilio Fiandrotti, and Marco Grangetto, "Loss-based sensitivity regularization: Towards deep sparse neural networks," *Neural Networks*, vol. 146, pp. 230–237, 2022.

[19] Bailin Li, Bowen Wu, Jiang Su, Guangrun Wang, and Liang Lin, "Eagleeye: Fast sub-net evaluation for efficient neural network pruning," 2020.

[20] Andrea Bragagnolo, Enzo Tartaglione, Attilio Fian-drotti, and Marco Grangetto, "On the role of structured pruning for neural network compression," in *2021 IEEE International Conference on Image Processing (ICIP)*, 2021, pp. 3527–3531.

[21] Enzo Tartaglione, Andrea Bragagnolo, and Marco Grangetto, "Pruning artificial neural networks: A way to find well-generalizing, high-entropy sharp minima," in *Artificial Neural Networks and Machine Learning – ICANN 2020*, Cham, 2020, pp. 67–78, Springer International Publishing.