ETH zürich

# Structured Sparsity for Efficiency in Model Inference

Semester Project

David Alexander Danhofer
ddanhofer@ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

**Supervisors:**
Dr. Peter Belcák
Prof. Dr. Roger Wattenhofer

July 2024

# Structured sparsity for efficiency in model inference

1st David Danhofer
*Department of Computer Science*
*ETH Zürich*
Zürich, Switzerland
ddanhofer@ethz.ch

***Abstract*—Repeated use of vision models for inference or deployment in resource-constrained environments necessitates effective acceleration techniques. This paper proposes a novel method to learn structured sparsity patterns enabling the utilization of hardware accelerations for regularly sparse matrix multiplications. The approach achieves an almost two-fold speed-up during inference without significantly affecting model performance while leaving the original model weights unchanged.**

***Index Terms*—computer vision, sparsity, efficiency**

## I. Introduction

With ever more models used in production settings it becomes increasingly interesting to look for ways to reduce resource spending over the model lifetime. This can be achieved by reducing the inference cost of the model by learning a modification of the model once before deployment reducing inference costs, i.e., a sparse masking of the weights. This way the dense matrix operations usually required can be replaced by cheaper and faster operations on sparse matrices. While many works have demonstrated that sparse (pruned) submodels can solve the same task at almost no loss of performance [7] the sparsity of the models does not necessarily have to adhere to a specific pattern making it difficult to leverage computational speedups. At the same time limiting the choices for these sparse patterns means the patterns need to be chosen carefully with the goal of minimizing the loss in inference performance in mind. This paper proposes a novel method of learning regularly sparse masking patterns for convolutions, a key building block for computer vision models, outperforming available heuristics and showing that regular sparsity masks can be learned without significant performance loss for computer vision classification tasks. The proposed method provides the additional advantage of not changing the original set of pretrained weights.

## II. Background

In the following two relevant aspects for the subsequent work will be introduced, i.e., the notion and utility of regular sparsity and approaches centred around pruning Convolutional Neural Networks (CNNs).

**N:M sparsity.** Regular sparsity to accelerate network inference has been introduced in [11] as *N:M-sparsity* requiring $N$ out of $M$ contiguous elements to be zero. Beyond the general case of N:M sparsity the work addresses the practically interesting special case of 2:4 sparsity in which exactly half of the weights are pruned as illustrated in Fig. 1. This setting enables hardware acceleration via NVIDIA sparse tensor cores available from the NVIDIA Ampere architecture on via the TensorRT v8.0 library [10]. Since half the elements are negligible the amount of data to load from memory is almost halved number with the number of flops needed to conduct an operation on the sparse matrix also scaling accordingly. The challenge in turning a dense matrix into a 2:4 sparse matrix, however, lies in selecting the most useful two of the four weights in each quadruple. To this end [11] proposes a permutation regime that allows for preserving the weights based on magnitude and assessing the found pattern via a scoring mechanism, i.e., the efficacy score. The functionality is available via NVIDIA's Apex library [9]. Notably, pruning via Apex requires finetuning the network again after pruning to achieve comparable inference performance to the dense network in computer vision tasks, e.g., classification [10], [11].

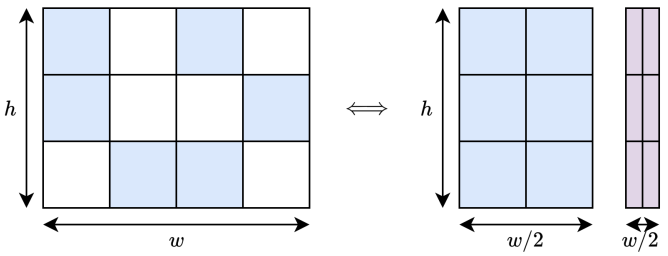


Fig. 1. A 2:4 sparse matrix of floating point values and its equivalent structured representation containing only the non-zero entries and 2-bit index values taking up roughly only half the space

**Pruning.** There have been different approaches to prune and this way accelerate CNNs including learned [14] and static approaches, e.g., based on weight magnitude [10], [15], methods that prune entire channels from a CNN [5], [14] and methods aimed at exploiting more fine-grained structures [3]. Ultimately, pruning can be used both as a regularizing mechanism as well as a method of finding a subnetwork less expensive at inference time while retaining the original performance.

The proposed method aims to specifically tackle the question of learning a regularly sparse subnetwork of a CNN substructure while leaving the original weights unchanged and preserving inference performance.

## III. Methods

In the following, first, the concept of modelling regular sparsity in a network architecture needed for leveraging

hardware acceleration is introduced then its application to convolutions in computer vision models is detailed out.

**Modelling regular sparsity.** As introduced in the previous section regular sparsity shall be leveraged in the form of 2:4´sparsity. There are exactly $\frac{4 \cdot 3}{2!} = 6$ ways of choosing two of the four elements in any quadruple. This can be modelled via a categorical variable $z$ with class probabilities $\pi_1, ..., \pi_6$ s.t. each probability denotes the probability of selecting the corresponding 2:4 sparsity pattern. Sampling from this distribution yields a 6-dimensional one-hot vector on the simplex $\Delta^5$ encoding this choice. Sampling from such a categorical distribution can be performed efficiently via the Gumbel-Max trick [2]

$$z = \text{onehot}(\arg\max_i [g_i + \log \pi_i]) \tag{1}$$

where $g_i \sim \text{Gumbel}(0, 1)$. Since the max operator is discrete and thus not differentiable, however, this method cannot be used directly in a neural network to be optimized via backpropagation. Instead a differentiable approximation is constructed by replacing the $\arg\max$ operator with a softmax. The choice vector $y$ can now be drawn as follows

$$y_i = \frac{\exp((g_i + \log \pi_i)/\tau)}{\sum_k \exp((g_k + \log \pi_k)/\tau)} \tag{2}$$

yielding a Gumbel-Softmax (GS) distribution [6] over the six choices additionally parametrized by the temperature parameter $\tau$. While not identical to a categorical distribution, the GS distribution approximates a categorical distribution over the choices for small temperature values, e.g., $\tau = 0.1$. This distribution allows for expressing the gradient as a deterministic function of the choice weights $\pi$ and an independent source of random noise and thus gradient propagation through the stochastic node [6]. By updating the class probabilities in the distribution in respect to the classification objective the choice weights can be optimized for the classification task and ideally the optimal choice is selected. An element-wise multiplication of the mask obtained this way yields the desired regularly sparse matrix.

**Regularly sparse convolutions.** A discrete two-dimensional convolution with a kernel $H \in \mathbb{R}^{c_{\text{in}} \times h \times w}$ convolves an input tensor $X \in \mathbb{R}^{c_{\text{in}} \times b \times d}$ into an output tensor $y \in \mathbb{R}^{b \times d}$ assuming zero padding. In the below formulation the functions $f$ and $g$ handle the padding for invalid combinations of input values, i.e., out of range values, else return the sum of the two input values:

$$y_{ij} = \sum_{c=1}^{c_{\text{in}}} \sum_{u=1}^{w} \sum_{s=1}^{h} H_{cus} X_{cf(i,u)g(j,s)} \tag{3}$$

Usually such a convolution is conducted with $c_{\text{out}}$ kernels to obtain an output $Y \in \mathbb{R}^{c_{\text{out}} \times b \times d}$. Alternatively, this convolution can also be expressed as a matrix multiplication between the same input $X \in \mathbb{R}^{c_{\text{in}} \times b \times d}$ and a weight matrix $W \in \mathbb{R}^{c_{\text{out}} \times (c_{\text{in}} wh)}$ constructed from the $c_{\text{out}}$ kernels in the convolutional layer:

$$\tilde{Y} = W \mathcal{U}(X) \tag{4}$$

The unfold operator $\mathcal{U}(\cdot)$ turns the matrix $X$ into a flattened matrix $\tilde{X} \in \mathbb{R}^{c_{\text{in}} wh \times L}$ where $L = (b + 2p_1 - w - 1)(d + 2p_2 - h - 1)$ denotes the number of blocks in the input. In the case of zero padding, i.e., full padding, the padding sizes in the respective dimensions for uneven kernel dimensions are $p_1 = \lfloor \frac{w}{2} \rfloor$ and $p_2 = \lfloor \frac{h}{2} \rfloor$ and thus $L = bd$. Reshaping $\tilde{Y}$ recovers the exact same $Y$ as in (3). To achieve a 2:4 sparse convolution compatible with the accelerated matrix multiplication for a 2:4 sparse matrix a masking matrix $M \in \{0, 1\}^{c_{\text{out}} \times c_{\text{in}} wh}$ whose (block) entries are sampled according to a GS distribution as described above is multiplied entry-wise to the weight matrix.

$$\tilde{Y} = (M \odot W)\, \mathcal{U}(X) \tag{5}$$

The corresponding masking layer therefore learns as many GS distributions as there are blocks of four elements in the matrix. Note that this assumes that the product $c_{\text{out}} \cdot c_{\text{in}} wh$ is a multiple of four, since $M$ can only contain a multiple of four entries to account for the size of the sparsity pattern. If this is not the case the matrix needs to be augmented column- or row-wise to contain a multiple of 4 entries. A schematic illustration of the two views on convolutions are illustrated in Fig. 2.

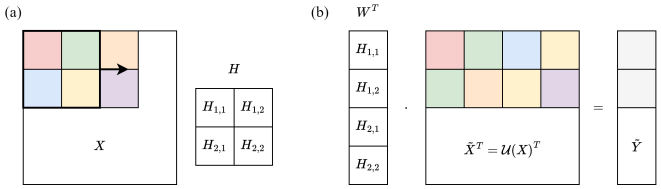


Fig. 2. Simplified visualizations of convolutions on single channel input $X$ of unspecified width and height and a single filter $H$ as (a) "standard" convolution with a moving filter and (b) as a matrix product between an unfolded input $\tilde{X}$ and a weight matrix $W$ derived from the filter

## IV. Results

**Architectures.** To evaluate whether the proposed performance gain via regular sparsity can be achieved without any significant loss in inference performance the following three architectures were considered: ResNet [4], VGG [13] and ShuffleNetV2 [8]. Each reformulated convolutional layer was equipped with a trainable masking layer masking the entries of the weight matrices in a 2:4 sparse fashion. The pretrained weights of the original architectures were copied into the modified models initializing the weights of the masking layers randomly using Glorot initialization [1] and a normal distribution with $\sigma = 10^{-6}$ for weights and biases respectively. Only the weights associated with the masking layers were configured to be trainable leaving the set of pretrained weights unchanged.

**Classification task.** The pretrained networks were retrained and evaluated on the multi-class classification challenge provided by the ImageNet dataset [12]. The data set contains approximately 1.2 million training images, 50k validation and 100k test images. To ensure consistency between the performance on the validation set and the reported accuracies on the test set the unmodified architectures were evaluated

showing no significant differences (cf. Table I). Likewise comparisons for the unmodified architectures and the modified architectures without masking were conducted showing no change in predictions and performance.

TABLE I
REPORTED AND VALIDATION CLASSIFICATION PERFORMANCE OF THE UNMODIFIED ARCHITECTURES MEASURED AS THE TOP-$k$ ACCURACY ON IMAGENET [12]

| Architecture | Parameters | reported | | validation | |
|---|---|---|---|---|---|
| | | top-1 | top-5 | top-1 | top-5 |
| ResNet-18 | 11.7M | 69.76 | 89.08 | 69.69 | 89.06 |
| ResNet-34 | 21.8M | 73.31 | 91.42 | 73.24 | 91.42 |
| ResNet-50 | 25.6M | 76.13 | 92.86 | 75.99 | 92.92 |
| VGG-16 | 138.2M | 71.59 | 90.38 | 73.47 | 91.49 |
| VGG-19 | 142.7M | 72.38 | 90.88 | 74.17 | 91.85 |
| ShuffleNetV2 2.0× | 7.4M | 76.23 | 93.00 | 76.24 | 92.91 |

**Training.** The pretrained modified architectures were trained according to a generic training procedure using Stochastic Gradient Descent (SGD) for optimization with a initial learning rate $\eta = 0.1$, a momentum of $\beta = 0.9$ and weight decay with a factor of $\lambda = 10^{-4}$. The learning rate was adjusted by a step scheduler with no warm-up period adjusting the learning rate every 30 epochs by a factor of $\gamma = 0.1$. The training did not make use of augmentation of the training data showing each sample randomly cropped in every epoch exactly once.

**Inference performance.** Table II summarizes the results obtained for the different architectures after the specified number of epochs of training. The results show little to no loss in performance for the ResNet architectures both in top-1 and top-5 accuracy after a short training period. In fact the performance even increases for the top-1 accuracy. For the VGG architecture on the other hand the performance worsened for both the top-1 and the top-5 accuracy noticeably dropping by as much as 6%. The tested ShuffleNet architecture converged to a non-performant level and both measures of accuracy worsened significantly. For all architectures convergence was reached quickly and further training did not affect the performance achieved.

TABLE II
VALIDATION CLASSIFICATION PERFORMANCE OF THE 2:4 SPARSE NETWORKS MEASURED AS THE TOP-$k$ ACCURACY ON IMAGENET [12] AND PERCENTAGE-WISE CHANGE TO THE REPORTED ACCURACY AFTER THE REFERENCE NUMBER OF EPOCHS TRAINED

| Architecture | 2:4 sparse | | |
|---|---|---|---|
| | top-1 | top-5 | epochs |
| ResNet-18 | 70.05 (+0.4%) | 88.16 (−1.0%) | 10 |
| ResNet-34 | 75.33 (+2.8%) | 91.18 (−0.3%) | 10 |
| ResNet-50 | 78.51 (+3.1%) | 92.84 (±0.0%) | 10 |
| VGG-16 | 67.79 (−5.3%) | 87.12 (−3.6%) | 12 |
| VGG-19 | 68.63 (−6.2%) | 87.63 (−3.6%) | 12 |
| ShuffleNetV2 2.0× | 11.00 (−85.6%) | 53.20 (−42.8%) | 12 |

To compare the results of the proposed method to a state of the art method of achieving a 2:4 sparse subnetwork two variants of the heuristic proposed in [11], the so-called efficacy score to evaluate permutations, available via NVIDIA's Apex library [9] were used in the same regime. To conform to the idea of not altering the original weights the networks were not retrained after pruning as originally proposed in [11] and the results are aggregated in Table III. It can be observed that for all variants of ResNet and VGG the loss in performance is significant even in the better performing variant allowing for channel permutations before selecting the 2:4 sparse subnetwork. The method proposed in this paper always manages to learn a significantly better sparse pattern. In the case of ShuffleNet, however, the efficacy score shows the existence of a 2:4 sparse subnetwork of far better performance than the one obtained by the proposed method. The failure to converge on ShuffleNet therefore shows a need for further research.

TABLE III
VALIDATION CLASSIFICATION PERFORMANCE OF THE 2:4 SPARSE NETWORKS OBTAINED VIA THE APEX LIBRARY [11] MEASURED AS THE TOP-$k$ ACCURACY ON IMAGENET [12]. THE NETWORKS ARE COMPARED IN TWO SETTINGS DISALLOWING AND ALLOWING PERMUTATIONS OF THE CHANNELS BEFORE PRUNING.

| Architecture | 2:4 sparse (Apex) | | | |
|---|---|---|---|---|
| | not permuted | | permuted | |
| | top-1 | top-5 | top-1 | top-5 |
| ResNet-18 | 17.20 | 36.13 | 21.48 | 41.76 |
| ResNet-34 | 43.75 | 68.27 | 49.27 | 73.71 |
| ResNet-50 | 30.09 | 52.52 | 48.37 | 72.47 |
| VGG-16 | 42.11 | 67.17 | 49.43 | 75.15 |
| VGG-19 | 58.99 | 82.16 | 63.99 | 85.64 |
| ShuffleNetV2 2.0× | 69.32 | 89.49 | 69.38 | 89.51 |

## V. DISCUSSION

The results show that it is possible to learn regular sparsity patterns that accelerate inference while not impeding classification performance with ShuffleNet being a notable outlier. However, the effects are not the same across all architectures. Specifically, the method appears to work better for ResNet-based architectures than VGG-based architectures. Within one family the change in performance does not necessarily correlate with the size of the models. Likewise the change in top-1 and top-5 accuracy is not consistently worse or better for one metric over the other exhibiting different effects for ResNets and VGGs. While clearly showing superior performance to the heuristic provided by the efficacy score [11] the case of ShuffleNet indicates that architectures can be difficult to learn a sparsity pattern for despite this pattern existing. It is conceivable that the training procedure significantly affects the obtained patterns and different approaches would need to be tried out for different model classes to verify this claim. Furthermore, the proposed method allows for exchanging the underlying weights against a new, updated, e.g., finetuned, weight set as it

does not rely on changing the weights. This only makes sense, however, if similar underlying weights yield similar sparsity patterns which is yet to be researched.

## VI. Conclusion

In this paper, a novel method for accelerating inference in computer vision architectures by a factor of close to $2\times$ at little to no performance loss by expressing convolutions as a 2:4 sparse matrix multiplication without changing the underlying weights was presented. It has been shown empirically that regular sparsity patterns can be learned that do not significantly affect performance thus effectively leveraging the widely available hardware acceleration provided by current hardware.

## References


[1] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks.

[2] E. J. Gumbel. *Statistical Theory of Extreme Values and Some Practical Applications: A Series of Lectures*, volume 33. US Government Printing Office, 1954.

[3] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.

[4] 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*, pages 770–778, 2016.

[5] Y. He, X. Zhang, and J. Sun. Channel Pruning for Accelerating Very Deep Neural Networks. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 1398–1406, Venice, Oct. 2017. IEEE.

[6] E. Jang, S. Gu, and B. Poole. Categorical Reparameterization with Gumbel-Softmax, Aug. 2017.

[7] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell. Rethinking the Value of Network Pruning, Mar. 2019.

[8] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun. ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design. In V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, editors, *Computer Vision – ECCV 2018*, volume 11218, pages 122–138. Springer International Publishing, Cham, 2018.

[9] {NVIDIA} Corporation. Apex (A PyTorch Extension) — Apex 0.1.0 documentation. https://nvidia.github.io/apex/, 2018.

[10] J. Pool, A. Sawarkar, and J. Rodge. Accelerating Inference with Sparsity Using the NVIDIA Ampere Architecture and NVIDIA TensorRT. https://developer.nvidia.com/blog/accelerating-inference-with-sparsity-using-ampere-and-tensorrt/, July 2021.

[11] J. Pool and C. Yu. Channel Permutations for N:M Sparsity. *Advances in neural information processing systems*, 34:13316–13327, 2021.

[12] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252, Dec. 2015.

[13] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition, Apr. 2015.

[14] Y. Wang, X. Zhang, X. Hu, B. Zhang, and H. Su. Dynamic Network Pruning with Interpretable Layerwise Channel Selection. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):6299–6306, Apr. 2020.

[15] T.-J. Yang, Y.-H. Chen, and V. Sze. Designing Energy-Efficient Convolutional Neural Networks Using Energy-Aware Pruning. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6071–6079, Honolulu, HI, July 2017. IEEE.