OPT2020: 12th Annual Workshop on Optimization for Machine Learning

Identifying Efficient Sub-networks using Mixed Integer Programming

Mostafa ElAraby MOUSTAFA.ELARABI@UMONTREAL.CA
Guy Wolf GUY.WOLF@UMONTREAL.CA
Margarida Carvalho CARVALHO@IRO.UMONTREAL.CA
Université de Montréal, Quebec, Canada

Abstract

We introduce a mixed integer program (MIP) for assigning importance scores to each neuron in deep neural network architectures which are guided by the impact of their simultaneous pruning on the main learning task of the network. By carefully devising the objective function of the MIP, we drive the solver to minimize the number of critical neurons (i.e., with high importance score) that need to be kept for maintaining the overall accuracy of the trained neural network. Further, the proposed formulation generalizes the recently considered lottery ticket optimization by identifying multiple "lucky" sub-networks resulting in optimized architecture that not only performs well on a single dataset, but also generalizes across multiple ones upon retraining of network weights. We demonstrate the ability of our formulation to prune neural networks with marginal loss in accuracy and generalizability on popular datasets and architectures. Finally, we present a scalable implementation of our method by decoupling the importance scores across layers using auxiliary networks.

Keywords: Deep learning, Pruning Neural Networks, Mixed-Integer Programming, Neurons Ranking, Combinatorial Optimization, Architecture Optimization.

1. Introduction

Deep learning has proven its power to solve complex tasks and to achieve state-of-the-art results in various domains such as image classification, speech recognition, machine translation, robotics and control [3, 18]. Over-parameterized deep neural models with more parameters than the training samples can be used to achieve state-of-the art results on various tasks [23, 31]. However, the large number of parameters comes at the expense of computational cost in terms of memory footprint, training time and inference time on resource-limited IOT devices [14, 20].

In this context, pruning neurons from an over-parameterized neural model has been an active research area. This remains a challenging open problem whose solution has the potential to increase computational efficiency and to uncover potential sub-networks that can be trained effectively. Neural Network pruning techniques [5, 11, 12, 15, 19, 24–27, 30] have been introduced to sparsify models without loss of accuracy. Most existing work focus on identifying redundant parameters and non-critical neurons to achieve a lossless sparsification of the neural model. The typical sparsification procedure includes training a neural model, then computing parameters importance and pruning existing ones using certain criteria, and fine-tuning the neural model to regain its lost accuracy. Existing pruning and ranking procedures are computationally expensive, requiring iterations of fine-tuning on the sparsified model and no experiments were conducted to check the generalization of sparsified models across different datasets.

© M. ElAraby, G. Wolf & M. Carvalho.

We remark that sparse neuron connectivity is often used by modern network architectures, and perhaps most notably in convolutional layers. Indeed, the limited size of the parameter space in such cases increases the effectiveness of network training and enables the learning of meaningful semantic features from the input images [9]. Inspired by the benefits of sparsity in such architecture designs, we aim to leverage the neuron sparsity achieved by our framework to attain optimized neural architectures that can generalize well across different datasets.

Contributions. In our proposed framework, we formalize the notation of *neuron importance* as a score between 0 and 1 for each neuron in a neural network and the associated dataset. The neuron importance score reflects how much activity decrease can be inflicted in it, while controlling the loss on the neural network model accuracy. Concretely, we propose a mixed integer programming formulation (MIP) that allows the computation of each fully connected's neuron and convolutional feature map importance score and that takes into account the error propagation between the different layers. In addition, we extend the proposed formulation to support convolutional layers computed as matrices multiplication using Toeplitz format [10] with an importance score associated with each feature map [21].

2. Preliminaries

Consider layer l of a trained ReLU neural network with $\boldsymbol{W^l}$ as the weight matrix, w_i^l row i of $\boldsymbol{W^l}$, and b^l the bias vector. For each input data point x, let h^l be a decision vector denoting the output value of layer l, i.e. $h^l = ReLU(\boldsymbol{W^l} h^{l-1} + b^l)$ for $l > 0$ and $h^0 = x$, and z_i^l be a binary variable taking value 1 if the unit i is active, i.e. $w_i^l h^{l-1} + b_i^l \geq 0$, and 0 otherwise. Finally, let L_i^l and U_i^l be constants indicating a valid lower and upper bound for the input of each neuron i in layer l. We discuss the computation of these bounds in Sec. 3.2. For now, we assume that L_i^l and U_i^l are sufficiently small and large numbers, respectively, i.e., the so-called Big-M values. Next, we provide the representation of ReLU neural networks of [7] called big-M formulation. Although, Anderson et al. [1] proposed an ideal MIP formulation, where there is an exponential number of facets defining constraints that can be separated efficiently, we used only the big-M formulation which performed well since we can compute tight local bounds. For sake of simplicity, we describe the formulation for one layer l of the model at neuron i and one input data point x:

$$h_i^0 = x_i \quad \text{if } l = 0, \text{ otherwise} \tag{1a}$$

$$h_i^l \geq 0, \tag{1b}$$

$$h_i^l + (1 - z_i^l) L_i^l \leq w_i^l h^{l-1} + b_i^l, \tag{1c}$$

$$h_i^l \leq z_i^l U_i^l, \tag{1d}$$

$$h_i^l \geq w_i^l h^{l-1} + b_i^l, \tag{1e}$$

$$z_i^l \in \{0, 1\}. \tag{1f}$$

In (1a), the initial decision vector h^0 is forced to be equal to the input x of the first layer. When z_i^l is 0, constraints (1b) and (1d) force h_i^l to be zero, reflecting a non-active neuron. If an entry of z_i^l is 1, then constraints (1c) and (1e) enforce h_i^l to be equal to $w_i^l h^{l-1} + b_i^l$. After formulating the ReLU, if we relax the binary constraint (1f) on z_i^l to $[0, 1]$, we obtain a linear programming problem which is easier and faster to solve. Furthermore, the quality (tightness) of such relaxation highly

depends on the choice of tight upper and lower bounds, U_i^l, L_i^l. In fact, the determination of tight bounds reduces the search space and hence, the solving time.

3. MIP formulation

In what follows, we adapt the MIP constraints (1) to quantify neuron importance, and we describe the computation of the bounds L_i^l and U_i^l. Our goal is to compute importance scores for all layers in the model in an integrated fashion, as Yu et al. [29] have shown to lead to better predictive accuracy than layer by layer.

3.1. ReLU layers

In ReLU activated layers, we keep the previously introduced binary variables z_i^l, and continuous variables h_i^l. Additionally, we create the continuous decision variables $s_i^l \in [0, 1]$ representing neuron i importance score in layer l. In this way, we modified the ReLU constraints (1) by adding the neuron importance decision variable s_i^l to constraints (1c) and (1e):

$$h_i^l + (1 - z_i^l)L_i^l \leq w_i^l h^{l-1} + b_i^l - (1 - s_i^l)\max(U_i^l, 0) \tag{2a}$$

$$h_i^l \geq w_i^l h^{l-1} + b_i^l - (1 - s_i^l)\max(U_i^l, 0). \tag{2b}$$

In (2), when neuron i is activated due to the input h^{l-1}, i.e. $z_i^l = 1$, h_i^l is equal to the right-hand-side of those constraints. This value can be directly decreased by reducing the neuron importance s_i^l. When neuron i is non-active, i.e. $z_i^l = 0$, constraint (2b) becomes irrelevant as its right-hand-side is negative. This fact together with constraints (1b) and (1d), imply that h_i^l is zero. Now, we claim that constraint (2a) allows s_i^l to be zero if that neuron is indeed non-important, i.e., for all possible input data points, neuron i is not activated. This claim can be shown through the following observations. Note that decisions h and z must be replicated for each input data point x as they present the propagation of x over the neural network. On the other hand, s evaluates the importance of each neuron for the main learning task and thus, it must be the same for all data input points. Thus, the key ingredients are the bounds L_i^l and U_i^l that are computed for each input data point, as explained in Sec. 3.2. In this way, if U_i^l is non-positive, s_i^l can be zero without interfering with the constraints (2). The latter is enforced by the objective function derived in Sec. 3.3. We note that this MIP formulation can naturally be extended to convolutional layers converted to matrix multiplication using toeplitz matrix [10] and with an importance score associated with each feature map. We refer the reader to the appendix for a detailed explanation.

3.2. Bounds Propagation

In the previous MIP formulation, we assumed a large upper bound U_i^l and a small lower bound L_i^l. However, using large bounds may lead to long computational times and a lost on the freedom to reduce the importance score as discussed above. In order to overcome these issues, we tailor these bounds accordingly with their respective input point x by considering small perturbations on its

value:

$$L^0 = x - \epsilon \tag{3a}$$
$$U^0 = x + \epsilon \tag{3b}$$
$$L^l = \boldsymbol{W}^{(l-)} U^{l-1} + \boldsymbol{W}^{(l+)} L^{l-1} \tag{3c}$$
$$U^l = \boldsymbol{W}^{(l+)} U^{l-1} + \boldsymbol{W}^{(l-)} L^{l-1} \tag{3d}$$
$$\boldsymbol{W}^{(l-)} \triangleq \min(\boldsymbol{W}^{(l)}, 0) \tag{3e}$$
$$\boldsymbol{W}^{(l+)} \triangleq \max(\boldsymbol{W}^{(l)}, 0). \tag{3f}$$

Propagating the initial bounds of the input data points throughout the trained model will create the desired bound using simple arithmetic interval [22]. The obtained bounds are tight, narrowing the space of feasible solutions.

3.3. Objective

The aim for the proposed framework is to sparsify non-critical neurons without reducing the predictive accuracy of the pruned ANN To this end, we combine two optimization objectives.

Our first objective is to maximize the set of neurons sparsified from the trained ANN. Let n be the number of layers, N^l the number of neurons at layer l, and $I^l = \sum_{i=1}^{N^l}(s_i^l - 2)$ be the sum of neuron importance scores at layer l with s_i^l scaled down to the range $[-2, -1]$.

In order to create a relation between neurons' importance score in different layers, our objective becomes the maximization on the amount of neurons sparsified from the $n-1$ layers with higher score I^l. Hence, we denote $A = \{I^l : l = 1, \ldots, n\}$ and formulate the sparsity loss as

$$\text{sparsity} = \frac{\max\limits_{A' \subset A, |A'| = (n-1)} \sum\limits_{I \in A'} I}{\sum_{l=1}^{n} |N^l|}. \tag{4}$$

Table 1: Importance of re-scaling sparsification objective to prune more neurons shown empirically on LeNet-5 model using threshold 0.05, by comparing accuracy on test set between reference model (Ref.), and pruned model (Masked).

DATASET	SCALING	REF. ACC.	MASKED ACC.	PRUNING PERCENTAGE (%)
MNIST	$s_i^l - 2$	$98.9\% \pm 0.1$	$\mathbf{98.7\% \pm 0.1}$	$\mathbf{13.2\% \pm 2.9}$
	$s_i^l - 1$		$98.8\% \pm 0.1$	$9.6\% \pm 1.1$
	s_i^l		$98.9\% \pm 0.2$	$8\% \pm 1.6$
FASHION-MNIST	$s_i^l - 2$	$89.9\% \pm 0.2$	$\mathbf{89.1\% \pm 0.3}$	$\mathbf{17.1\% \pm 1.2}$
	$s_i^l - 1$		$89.2\% \pm 0.1$	$17\% \pm 3.4$
	s_i^l		$89\% \pm 0.4$	$10.8\% \pm 2.1$

Here, the objective is to maximize the number of non-critical neurons at each layer compared to other layers in the trained neural model. Note that only the $n-1$ layers with the largest importance score will weight in the objective, allowing to reduce the pruning effort on some layer that will naturally have low scores. The sparsity quantification is then normalized by the total number of neurons.

In Table 1, we compare re-scaling the neuron importance score in the objective function to $[-2,-1]$, to $[-1,0]$ and no re-scaling $[0,1]$ using LeNet-5 [16] trained on MNIST [17] and Fashion-MNIST [28]. This comparison shows empirically the importance of re-scaling the neuron importance score to optimize sparsification through neuron pruning.

Our second objective is to minimize the loss of *important* information due to the sparsification of the trained neural model. Additionally, we aim for this minimization to be done without relying on the values of the logits, which are closely correlated with neurons pruned at each layer. To that end, we formulate this optimization objective using the marginal softmax as proposed in [8]. Using marginal softmax allows the solver to focus on minimizing the misclassification error without relying on logit values. Formally, we write the objective

$$\text{softmax} = \sum_{i=1}^{N^n} \log\left[\sum_{c} \exp(h_{i,c}^n)\right] - \sum_{i=1}^{N^n}\sum_{c} Y_{i,c} h_{i,c}^n, \tag{5}$$

where index c stands for the class label. The used marginal softmax objective keeps the correct predictions of the trained model for the input batch of images x having one hot encoded labels Y without considering the logit value.

Finally, we combine the two objectives to formulate the multi-objective loss

$$\text{loss} = \text{sparsity} + \lambda \cdot \text{softmax} \tag{6}$$

as a weighted sum of sparsification regularizer and marginal softmax, as proposed by Ehrgott [6]. Our experiments revealed that $\lambda = 5$ generally provides the right trade-off between our two objectives; see the appendix for experiments with value of λ.

4. Experiments

In this section, we validate our proposed approach in revealing efficient sub-networks from ANN architectures. We introduce the steps used to generate the efficient sub-network using our framework in Algorithm 1.

Algorithm 1: Optimizing ANN Architectures using a MIP.

Result: Efficient sub-network
Input: Trained ANN, dataset D, threshold.
Step 1: Select subset of images $D' \subset D$ to be fed into the MIP.
Step 2: Solve MIP restricted to D' and save the vector of neuron importance scores s.
Step 3: Remove every neuron i from layer l with $s_i^l \leq$ threshold from ANN resulting in an efficient sub-network.

Table 2 shows that pruning non-critical neurons results in marginal loss and gives better performance. On the other hand, we observe a significant drop on the test accuracy when critical or a random set of neurons are removed compared with the reference model. If we fine-tune for just 1 epoch the sub-network obtained through our method, the model's accuracy can surpass the reference

Table 2: Pruning results on convolutional (Lenet-5, VGG-16) network architectures using three different datasets. We compare the test accuracy between the unpruned reference network (REF.), randomly pruned model (RP.), model pruned based on critical neurons selected by the MIP (CP.) and our non-critical pruning approach with (OURS + FT) and without (OURS) fine-tuning for 1 epoch. We used $\lambda = 5$ in the MIP objective

Dataset	MNIST	Fashion-MNIST	CIFAR-10	
Architecture	LeNet-5			VGG-16
Ref.	$98.9\% \pm 0.1$	$89.7\% \pm 0.2$	$72.2\% \pm 0.2$	$83.9\% \pm 0.4$
RP.	$56.9\% \pm 36.2$	$33\% \pm 24.3$	$50.1\% \pm 5.6$	$85\% \pm 0.4$
CP.	$38.6\% \pm 40.8$	$28.6\% \pm 26.3$	$27.5\% \pm 1.7$	$83.3\% \pm 0.3$
Ours	$\mathbf{98.7\% \pm 0.1}$	$\mathbf{87.7\% \pm 2.2}$	$\mathbf{67.7\% \pm 2.2}$	N/A
Ours + ft	$\mathbf{98.9\% \pm 0.04}$	$\mathbf{89.8\% \pm 0.4}$	$\mathbf{68.6\% \pm 1.4}$	$\mathbf{85.3\% \pm 0.2}$
Prune (%)	$17.2\% \pm 2.4$	$17.8\% \pm 2.1$	$9.9\% \pm 1.4$	$36\% \pm 1.1$
threshold	0.2	0.2	0.3	0.3

model. This is due to the fact that the MIP, while computing neuron scores, is solving its marginal softmax (5) on true labels.

Table 3 displays our experiments and respective results. When we compare generalization results to pruning using our approach on Fashion-MNIST and CIFAR-10, we discover that computing the critical sub-network LeNet-5 architecture on MNIST, is creating a more sparse sub-network with test accuracy better than zero-shot pruning without fine-tuning using our approach, and comparable accuracy with the original ANN.

Table 3: Cross-dataset generalization from dataset d_1 to dataset d_2. Test accuracies are presented for masked and unmasked (REF.) networks on d_2, as well as pruning percentage.

MODEL	SOURCE DATASET d_1	TARGET DATASET d_2	REF. ACC.	MASKED ACC.	PRUNING (%)
LENET-5	MNIST	FASHION MNIST	$89.7\% \pm 0.3$	$89.2\% \pm 0.5$	$16.2\% \pm 0.2$
		CIFAR-10	$72.2\% \pm 0.2$	$68.1\% \pm 2.5$	
VGG-16	CIFAR-10	MNIST	$99.1\% \pm 0.1$	$99.4\% \pm 0.1$	$36\% \pm 1.1$
		FASHION-MNIST	$92.3\% \pm 0.4$	$92.1\% \pm 0.6$	

5. Conclusion

We proposed a mixed integer program to compute neuron importance scores in ReLU-based deep neural networks. Our contributions focus on providing scalable computation of importance scores in fully connected and convolutional layers. We presented results showing these scores can be effectively used to prune unimportant parts of the network without significantly affecting its main task (e.g., showing small or negligible drop in classification accuracy). Further, our results indicate this approach allows automatic construction of efficient sub-networks that can be transferred and retrained on different datasets. The presented model introduces one of the first steps in understanding which components in a neural network are critical for its model capacity to perform a given task, which can have further impact in future work beyond the pruning applications presented here.

Acknowledgement

This work was partially funded by: IVADO (l'institut de valorisation des données) [*G.W.*, *M.C.*]; NIH grant R01GM135929 [*G.W.*]; FRQ-IVADO Research Chair in Data Science for Combinatorial Game Theory, and NSERC grant 2019-04557 [*M.C.*].

References

[1] Ross Anderson, Joey Huchette, Christian Tjandraatmadja, and Juan Pablo Vielma. Strong mixed-integer programming formulations for trained neural networks. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 27–42. Springer, 2019.

[2] Eugene Belilovsky, Michael Eickenberg, and Edouard Oyallon. Decoupled greedy learning of CNNs. *arXiv preprint arXiv:1901.08164*, 2019.

[3] Yoshua Bengio, Ian Goodfellow, and Aaron Courville. *Deep learning*, volume 1. Citeseer, 2017.

[4] Tom Brosch and Roger Tam. Efficient training of convolutional deep belief networks in the frequency domain for application to high-resolution 2D and 3D images. *Neural computation*, 27(1):211–227, 2015.

[5] Xin Dong, Shangyu Chen, and Sinno Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In *Advances in Neural Information Processing Systems*, pages 4857–4867, 2017.

[6] Matthias Ehrgott. *Multicriteria optimization*, volume 491. Springer Science & Business Media, 2005.

[7] Matteo Fischetti and Jason Jo. Deep neural networks and mixed integer linear optimization. *Constraints*, 23(3):296–309, 2018.

[8] Kevin Gimpel and Noah A Smith. Softmax-margin crfs: Training log-linear models with cost functions. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 733–736. Association for Computational Linguistics, 2010.

[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[10] Robert M Gray. Toeplitz and circulant matrices: A review, 2002. *URL http://ee. stanford. edu/˜ gray/toeplitz. pdf*, 2000.

[11] 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*, pages 1135–1143, 2015.

[12] Babak Hassibi, David G Stork, and Gregory J Wolff. Optimal brain surgeon and general network pruning. In *IEEE international conference on neural networks*, pages 293–299. IEEE, 1993.

[13] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[14] Nicholas D Lane, Sourav Bhattacharya, Petko Georgiev, Claudio Forlivesi, and Fahim Kawsar. An early resource characterization of deep learning on wearables, smartphones and internet-of-things devices. In *Proceedings of the 2015 international workshop on internet of things towards applications*, pages 7–12, 2015.

[15] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.

[16] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[17] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, 2:18, 2010.

[18] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[19] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340*, 2018.

[20] He Li, Kaoru Ota, and Mianxiong Dong. Learning iot in edge: Deep learning for the internet of things with edge computing. *IEEE network*, 32(1):96–101, 2018.

[21] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016.

[22] Ramon E Moore, R Baker Kearfott, and Michael J Cloud. *Introduction to interval analysis*, volume 110. Siam, 2009.

[23] Behnam Neyshabur, Zhiyuan Li, Srinadh Bhojanapalli, Yann LeCun, and Nathan Srebro. Towards understanding the role of over-parametrization in generalization of neural networks. *arXiv preprint arXiv:1805.12076*, 2018.

[24] Abdullah Salama, Oleksiy Ostapenko, Tassilo Klein, and Moin Nabi. Pruning at a glance: Global neural pruning for model compression. *arXiv preprint arXiv:1912.00200*, 2019.

[25] Thiago Serra, Abhinav Kumar, and Srikumar Ramalingam. Lossless compression of deep neural networks. *arXiv preprint arXiv:2001.00218*, 2020.

[26] Suraj Srinivas and R Venkatesh Babu. Data-free parameter pruning for deep neural networks. *arXiv preprint arXiv:1507.06149*, 2015.

[27] Chaoqi Wang, Roger Grosse, Sanja Fidler, and Guodong Zhang. Eigendamage: Structured pruning in the kronecker-factored eigenbasis. *arXiv preprint arXiv:1905.05934*, 2019.

[28] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[29] Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis. Nisp: Pruning networks using neuron importance score propagation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9194–9203, 2018.

[30] Wenyuan Zeng and Raquel Urtasun. Mlprune: Multi-layer pruning for automated neural network compression. In *International Conference on Learning Representations (ICLR)*, 2018.

[31] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

Appendix A. MIP formulations

In Appendix A.1, details on the MIP constraints for convolutional layers are provided. Appendix A.2 explains the formulation used to represent pooling layers.

A.1. MIP for convolutional layers

We convert the convolutional feature map to a Toeplitz matrix and the input image to a vector. This allow us to use simple matrix multiplication which is computationally efficient. Moreover, we can represent the convolutional layer using the same formulation of fully connected layers presented in Sec. 3.1.

Toeplitz Matrix is a matrix in which each value is along the main diagonal and sub diagonals are constant. So given a sequence a_n, we can create a Toeplitz matrix by putting the sequence in the first column of the matrix and then shifting it by one entry in the following columns:

$$\begin{pmatrix} a_0 & a_{-1} & a_{-2} & \cdots & \cdots & \cdots & \cdots & a_{-(N-1)} \\ a_1 & a_0 & a_{-1} & a_{-2} & & & & \vdots \\ a_2 & a_1 & a_0 & a_{-1} & \ddots & & & \vdots \\ \vdots & a_2 & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & a_{-2} & \vdots \\ \vdots & & & \ddots & a_1 & a_0 & a_{-1} & a_{-2} \\ \vdots & & & & a_2 & a_1 & a_0 & a_{-1} \\ a_{(N-1)} & \cdots & \cdots & \cdots & \cdots & a_2 & a_1 & a_0 \end{pmatrix}. \tag{7}$$

Feature maps are flipped and then converted to a matrix. The computed matrix when multiplied by the vectorized input image will compute the fully convolutional output. For padded convolution we use only parts of the output of the full convolution, for strided convolutions we used sum of 1 strided convolution as proposed by Brosch and Tam [4]. First, we pad zeros to the top and right of the input feature map to become same size as the output of the full convolution. Next, we create a Toeplitz matrix for each row of the zero padded feature map. Finally, we arrange these small Toeplitz matrices in a big doubly blocked Toeplitz matrix. Each small Toeplitz matrix is arranged in the doubly Toeplitz matrix in the same way a Toeplitz matrix is created from input sequence with each small matrix as an element of the sequence.

A.2. Pooling Layers

We represent both average and max pooling on multi-input units in our MIP formulation. Pooling layers are used to reduce spatial representation of input image by applying an arithmetic operation on each feature map of the previous layer.

Avg Pooling layer applies the average operation on each feature map of the previous layer. This operation is linear and thus, it can directly be included in the MIP constraints:

$$h^{l+1} = \text{AvgPool}(h_1^l, \cdots, h_{N^l}^l) = \frac{1}{N^l} \sum_{i=1}^{N^l} h_i^l. \tag{8}$$

Max Pooling takes the maximum of each feature map of the previous layer:

$$h^{l+1} = \text{MaxPool}(h_1^l, \cdots, h_{N^l}^l) = \max\{h_1^l, \cdots, h_{N^l}^l\}. \tag{9}$$

This operation can be expressed by introducing a set of binary variables $m_1, \cdots, m_{N^l}$, where $m_i = 1$ implies $x = \text{MaxPool}(h_1^l, \cdots, h_{N^l}^l)$:

$$\sum_{i=1}^{N^l} m_i = 1 \tag{10a}$$

$$\left.\begin{aligned} &x \geq h_i^l, \\ &x \leq h_i^l m_i + U_i(1 - m_i) \\ &m_i \in \{0, 1\} \end{aligned}\right\} i = 1, \cdots, N^l. \tag{10b}$$

Appendix B. Scalability improvements

In Appendix B.1, we describe a methodology that aims at speed-up the computation of neuron importance scores by relying on decoupled greedy learning.

B.1. Decoupled Greedy Learning

We use decoupled greedy learning [2] to parallelize learning of each layer by computing its gradients and using an auxiliary network attached to it. By using this procedure, we have auxiliary networks of the deep neural network that represent subsets of layers thus allowing us to solve the MIP in sub-representations of the neural network.

Training procedure We start by constructing auxiliary networks for each convolutional layer except the last convolutional layer that will be attached to the classifier part of the model. During the training each auxiliary network is optimized with a separate optimizer and the auxiliary network's output is used to predict the back-propagated gradients. Each sub-network's input is the output of the previous sub-network and the gradients will flow through only the current sub-network. In order to parallelize this operation a replay buffer of previous representations should be used to avoid waiting for output from previous sub-networks during training.

Auxiliary Network Architecture We use a spatial averaging operation to construct a scalable auxiliary network applied to the output of the trained layer and to reduce the spatial resolution by a factor of 4, then applying one 1×1 convolution with batchnorm [13] followed by a reduction to 2×2 and a one-layer MLP. The architecture used for the auxiliary network is smaller than the one mentioned in the paper leading to speed up the MIP solving time per layer.

MIP representation After training each sub-network, we create a separate MIP formulation for each auxiliary network using its trained parameters and taking as input the output of the previous sub-network. This operation can be easily parallelized and each sub-network can be solved independently. Then, we take the computed neuron importance scores per layer and apply them to the main deep neural network. Since these layers were solved independently, we fine tune the network for one epoch to back-propagate the error across the network. The created sub-network can be generalized across different datasets and yields marginal loss.