Internet of Things 20 (2022) 100599

Contents lists available at ScienceDirect

Internet of Things

journal homepage: www.elsevier.com/locate/iot

Research article

Resource efficient AI: Exploring neural network pruning for task specialization

Dieter Balemans [a,b,*], Philippe Reiter [a], Jan Steckel [b,c], Peter Hellinckx [a]

[a] Faculty of Applied Engineering - IDLab, University of Antwerp - imec, Sint-Pietersvliet 7, 2000 Antwerp, Belgium
[b] Faculty of Applied Engineering - CoSysLab, University of Antwerp, Groenenborgerlaan 171, 2020 Antwerp, Belgium
[c] Flanders Make Strategic Research Centre, Oude Diestersebaan 133, 3920 Lommel, Belgium

ARTICLE INFO

Keywords:
Neural network compression
Machine learning
Explainable AI
Neural network pruning
Edge inference

ABSTRACT

This paper explores the use of neural network pruning for transfer learning applications for more resource-efficient inference. The goal is to focus and optimize a neural network on a smaller specialized target task. With the advent of IoT, we have seen an immense increase in AI-based applications on mobile and embedded devices, such as wearables and other smart appliances. However, with the ever-increasing complexity and capabilities of machine learning algorithms, this push to the edge has led to new challenges due to the constraints imposed by the limited availability of resources on these devices. Some form of compression is needed to allow for state-of-the-art convolutional neural networks to run on edge devices. In this work, we adapt existing neural network pruning methods to allow them to specialize networks to only focus on a subset of what they were originally trained for. This is a transfer learning use-case where we optimize large pre-trained networks. This differs from standard optimization techniques by allowing the network to forget certain concepts and allow the network's footprint to be even smaller. We compare different pruning criteria, including one from the field of Explainable AI (XAI), to determine which technique allows for the smallest possible network while maintaining high performance on the target task. Our results show the benefits of using network specialization when executing neural networks on embedded devices both with and without GPU acceleration.

1. Introduction

In state-of-the-practice IoT platforms, complex decisions are generally made in the cloud rather than on the edge devices. Data from each sensor is sent to the cloud, and based on this, a decision is sent back to the actuators on the edge. This centralized setup can lead to problems, for example, for real-time applications where variable communication latency can result in failure to meet a deadline. Therefore, we deem it necessary to push the decision-making process closer to the edge where the sensors and actuators are located. This, however, leads to new challenges due to constraints imposed by the limited availability of resources (e.g., computing power, memory, energy) on the edge devices.

In many state-of-the-art applications, Deep Convolutional Neural Networks (CNNs) are being used for a variety of applications. CNNs have proven to possess incredible predictive power in many application domains. They are being heavily used in computer vision tasks [1–3], natural language processing [4], speech recognition [5], and many more. In many IoT applications, these networks are not executed on edge devices as these networks are too expensive both storage-wise and computationally. CNN's and other

* Corresponding author at: Faculty of Applied Engineering - IDLab, University of Antwerp - imec, Sint-Pietersvliet 7, 2000 Antwerp, Belgium.
E-mail addresses: dieter.balemans@uantwerpen.be (D. Balemans), philippe.reiter@uantwerpen.be (P. Reiter), jan.steckel@uantwerpen.be (J. Steckel), peter.hellinckx@uantwerpen.be (P. Hellinckx).

https://doi.org/10.1016/j.iot.2022.100599
Received 15 June 2022; Received in revised form 11 August 2022; Accepted 11 August 2022
Available online 17 August 2022

architectures (for example, transformer networks [4]) consist of billions of parameters and Floating-Point Operations (FLOPs), which are not feasible for edge devices to compute within required deadlines. Therefore, in order to push the decision-making to the edge, compression of these networks is needed.

Recently there has been an increasing interest in AI component adaptation. This includes making components of an AI application adjustable to specific environmental characteristics and constraints. Therefore, we investigate the feasibility of pruning techniques for network subtask specialization. This involves optimizing a pre-trained network specifically for a subset of its original trained task. We call this network specialization. This way, we can save resources by only focusing on the subset for a particular environment. As an example, consider object detection or classification models. These models detect a range of classes with a specific accuracy for each class. However, these accuracy levels might not always be needed in all application contexts. It might be favorable to reduce the accuracy in favor of resource gain—for example, a visual classification task, where a model is trained to classify traffic signs. We could optimize this model for highway operation by abstracting the types of signs (rural roadway signage, for instance) it still needs to classify accurately. Making these abstractions will lower the classification accuracy of certain output classes; however, the model will be able to be executed with fewer resources and can be evaluated faster, which is beneficial on highways where fast response times are essential. Furthermore, we can create multiple optimized versions of an off-the-shelve pre-trained network, each specialized for a different subset. This allows us to optimally switch between the optimized versions in order to save resources in dynamic environments. This way, we can have a version of the network specialized for people detection that is used in a city center and a version specialized for cars and traffic signs for highways.

In order to generate these versions, we adapt existing neural network pruning methods to allow them to specialize networks to only focus on a subset of what they were initially trained for. Large pre-trained models are widely and publicly available. With the methods we use in this work, these networks can be optimized and specialized to match a user's application needs without the need for training a network from scratch. This work compares different pruning criteria in a structured, iterative layerwise pruning strategy. The main goal is to find the smallest possible network while maintaining high performance on the target task. By smallest possible network, we mean a network consisting of the least amount of parameters and requiring the least amount of FLOPs to evaluate. In Section 4, we explain how we accomplish network specialization using neural network pruning. It is important to note that there is almost always a trade-off between the number of parameters and the FLOPs. In a neural network, the bulk of the parameters are in the fully connected layers, while the bulk of FLOPs are located in the convolutional layers. The priority of reducing the number of parameters or FLOPs should be updated depending on the target application and available hardware.

2. Related work

Neural network compression is not a new field. There are numerous methods for accelerating and shrinking a neural network. Techniques like pruning, weight sharing [6,7], quantization [8] and knowledge distillation [9] have proven to be very effective. Furthermore, the lottery ticket hypothesis [10] reveals that training large overparameterized networks makes it easier to find subnetworks that, when trained from scratch, can match the large network's performance. This also points to the advantages of network compression over training smaller networks. In this work, we focus on neural network pruning performed after training for compressing and specializing neural networks.

Neural network pruning has been around since the beginning of deep learning. In general, neural network pruning consists of removing irrelevant parameters from the network. One of the simplest methods from Hagiwara [11] is to use a threshold γ that dictates which weights or nodes are removed. Since then, many other pruning methods have been proposed. LeCun et al. [12] proposed a pruning method based on the second-order derivatives in their paper about optimal brain damage. This led the way to other pruning methods based on weights [13,14], activations [15], and gradients [16–18] in the network. For instance, Lee et al. [19] use a saliency-based criterion consisting of the lowest absolute value of the normalized gradient multiplied by the weight magnitude for a given set of mini-batch inputs.

In this paper, we focus on pruning for the specialization of the network. The goal is to achieve smaller and faster networks for inference by removing unnecessary trained concepts from the network. This comes down to localizing knowledge concepts inside a network and using these locations to propose optimization. This localization is related to the field of explainable and interpretable AI (XAI). This also inspired us to use an XAI-based pruning criterion. More specifically, in this work, we use Layer-wise Relevance Propagation (LRP) [20]. LRP is a saliency method that uses decomposition to calculate the most relevant pixels of the input and thus is capable of highlighting the most relevant parts of a network for a given task. The pruning method that we use is inspired by the work of Yeom et al. [21]. In that work, the authors propose to use LRP to guide the network pruning. Their results focus on the classical compression use-case of preserving all knowledge in the network. In Balemans et al. [22] the authors also propose to use LRP as guide for pruning. Molchanov et al. [23] proposed a similar criterion based on a Taylor expansion. They propose an iterative approach in their work, however, which can be very time-consuming. Finally, Lee et al. [24] also propose a saliency-based method, which is data-driven. This alleviates the dependency on weights and enables them to prune the network before training.

Neural network specialization can be seen as a form of transfer learning as we start from an original trained model and only want to extract the necessary knowledge for the specialization. Transfer learning using pruning can be found in literature as well. Lui et al. [25] propose a similar pipeline as in our work and use target-aware weight importance to guide the pruning. Some applied research can also be found [26,27]. This seems to point to the interest from both research and industry for smaller, faster networks that can be adapted and optimized from off-the-shelve, general trained networks. In our work, we focus on the subtask specialization for AI component adaptation as explained in Section 1. Finally, we optimize networks to run on edge devices. In this area, there have been other advances as well. Alippi et al. [28] introduce a methodology for the design and porting of CNNs to embedded systems limited in resources. They use approximation techniques to reduce the computational load. In Section 5.5, we show the results of our method on embedded platforms both with and without GPU acceleration.

3. Contributions

Our main contributions in this paper include:

1. The introduction of the network specialization use-case. A form of transfer learning by compressing the network and optimizing it for a specific subtask. This has great potential for making AI applications context- and resource-aware. Specialization pruning in combination with intelligent switching between different optimized versions can reduce the computational load of an AI application substantially.
2. The use of an Explainable AI pruning criterion, i.e., LRP, for calculating importance scores for the neurons of a neural network. This allows for learned concepts inside the network to be located and serves as a guide for pruning. This is explained in Section 4.
3. We show comparative results of a commonly-used, structured iterative layerwise pruning schedule in combination with different pruning criteria. This can be found in Sections 4.1 and 5.

4. Neural network pruning for specialization

In this section, we discuss the proposed pruning strategy and its variations. A pruning strategy is an algorithm for finding the optimal pruning rate for the network for which the impact on the performance of the network on the target task is minimal. For building a pruning strategy, there are several considerations:

1. **Structured and unstructured pruning**. This has implications on which structures we remove from the network. In structured pruning, we remove entire 'block'-like structures from the network, i.e., filters or entire neurons. In unstructured pruning, connections are removed individually. Structured pruning allows for the network to become physically smaller but has a bigger impact on the network's performance. In this research, we use a structured pruning method as it has a larger positive impact on today's hardware. Structured pruning directly alters the network's architecture; therefore, fewer resources are consumed. Later, we discuss the details of structured pruning for fully connected and convolutional layers.
2. **Iterative retraining versus one-shot pruning**. A common practice in pruning is to perform the removal of the least important connections iteratively, accompanied by a fine-tuning (retraining) step. This allows for preserving the network accuracy while pruning. However, this can be very time-consuming and not always necessary, as the accuracy can drop for certain outputs when specializing the networks. Therefore, in this research, we look at pruning without fine-tuning (one-shot), as well. This way, we can optimize networks quickly and with fewer resources.
3. **Parameter attribution**. In literature, this is also called a pruning criterion. In order for a pruning algorithm to select which parts of the network it can safely remove, the pruning algorithm needs to be able to rank the parameters of the network by their contribution to the target task. As mentioned earlier, our goal is to remove concepts from the network that are not needed for a specifically selected target task. In order to accomplish this, we introduce an attribution method from the field of XAI, called Layerwise Relevance Propagation [20]. We compare this to more traditional methods like weight-based [29], activation based [30] and random attribution.

With these considerations in mind, we propose to use a structured layerwise iterative pruning strategy [31]. Layerwise means we prune the network such that only one layer is considered for each iteration. This makes the strategy very flexible, as we can precisely tune the network to our (computational) needs. However, this also means we need to find a specific optimal pruning rate for each layer. We tackle this in an iterative way by pruning a layer for $x\%$ iteratively, checking each iteration if a stopping threshold has been reached. The stopping threshold can, for instance, be the test accuracy of the network dropping below a certain value or a maximum computational cost being reached. We start at the last layer of the network and work our way back to the start. After each iteration, the attribution scores are recalculated in order to make sure the scores are redistributed over the remaining nodes. This strategy can be used in two variations: The first one is pruning without fine-tuning; the second one is with fine-tuning. As mentioned earlier, this is the difference between iterative retraining and one-shot pruning. The above-described algorithm can be found in Algorithm 1.

4.1. Structured pruning

As mentioned earlier, we remove parameters in a structured manner based on attribution scores. This means locating block-like structures of parameters that can be removed and thus change the network architecture. Currently, we target networks consisting of fully connected layers and convolutional layers. In these layers, structured pruning is performed on neuron/filter level. It is important to note that removing a neuron/filter in one layer does not only impact that layer, as removing a neuron/filter changes the dimensions of the layer. Therefore, the subsequent layers also need to be adapted in order for the network to be still valid. In this work, the network is optimized after attribution by removing zero neurons/filters while making sure the output dimensions still match the subsequent layers input dimensions. In the following subsections, we describe the pruning procedures for fully connected and convolutional layers.

Data: Model: M, Pruning Rate Step: p
Result: Compressed Model: M'
for *layers* l *of* M **do**
  $x = 0$;
  while *(accuracy > threshold) and* ($x <$ *maximum sparsity)* **do**
    calculate importance scores;
    if l *is convolution layer* **then**
      calculate filter importance;
    end
    remove x% lowest neurons/filters;
    $x = x + p$;
  end
  if *not one-shot* **then**
    fine-tune M;
  end
end

Algorithm 1: Structured Layerwise Iterative Pruning Algorithm

4.1.1. Fully-connected layers

Consider a fully-connected layer with:

- Weight matrix $W(k, l)$ where k is the number of output features of the layer, and l the number of input features
- Attribution matrix $R(N, k)$ where N is the number of samples used in the evaluation dataset (or batchsize) and k is the number of output features. R contains k attribution scores r for each of the N samples. Depending on the configuration, the attribution scores are averaged or the maximum value is taken of all the samples:
  - $R_{avg} = \frac{\sum_i^N r_i}{N}$ which gives a vector with k elements.
  - $R_{max} = max_{j \in N}(r_{ij})\ \forall i \in k$ which gives a vector with k elements.

Based on the attribution matrix R_{avg} or R_{max}, a pruning mask is generated by selecting the x%-percent lowest-scoring nodes for removal from the layer. Here we are removing all input and output connections to a node from the layer.

4.1.2. Convolutional layers

Consider a convolutional layer with:

- Weight matrix $W(C_{out}, C_{in}, k_w, k_h)$ where C_{out} is the number of output channels, C_{in} is the number of input channels, and k_w, k_h is the kernel width and height.
- Attribution matrix $R(N, C_{out}, W_{out}, H_{out})$ where N is the number of samples used in the evaluation dataset (or batchsize), C_{out} is the number of output channels, W_{out} and H_{out} are the width and height of the output tensor of each channel in the layer. To acquire an attribution score for each filter, the attribution scores r_{ij} of all indices of each the channel are summed: $R_{filter} = \sum_{j=1}^{W_{out}} \sum_{i=1}^{H_{out}} r_{ij}$

This gives R_{filter} with dimensions $(N, C_{out}, 1)$. The next step is the same as for the fully connected layers: averaging or taking the maximum of all samples N, and thus ending up with a matrix with dimensions $(C_{out}, 1)$. These scores are used for pruning by ranking them and selecting the x%-percent lowest-scoring filters for removal from the layer.

4.1.3. ResNet considerations

In this work, we also perform structured pruning of residual networks (ResNets). These networks feature modular blocks and skip connections, representing commonly used modern networks. In order to prune these networks, we employ a similar method as proposed by Crowley et al. [32]. For a standard residual block, they do not change the number of in- and output channels; however, the intermediate channels are pruned. This enables the removal of filters in the first convolution and decreases the filter channel size in the second convolution, all the while making sure the dimensions for the skip connection operations remain unchanged.

4.2. Pruning for specialization

In order to determine the parameter importance scores of the network, we employ a three-step method. An overview of our parameter attribution method is shown in Fig. 1. The first step involves a forward pass of the original network using an evaluation dataset. The internal activations of the network for this particular dataset are stored, and thus, this dataset determines for which task the network is optimized. After the processing of the evaluation dataset, an attribution score for each node is calculated during a backward pass using one of the pruning criteria. We compare a l_1-norm weight magnitude pruning criterion from Li et al. [29], an activation pruning criterion [30], a LRP based criterion [21], and a random pruning criterion. This process can be done in batches of the dataset. It is important to note that this method is very efficient and fast as it only requires two passes through the network.

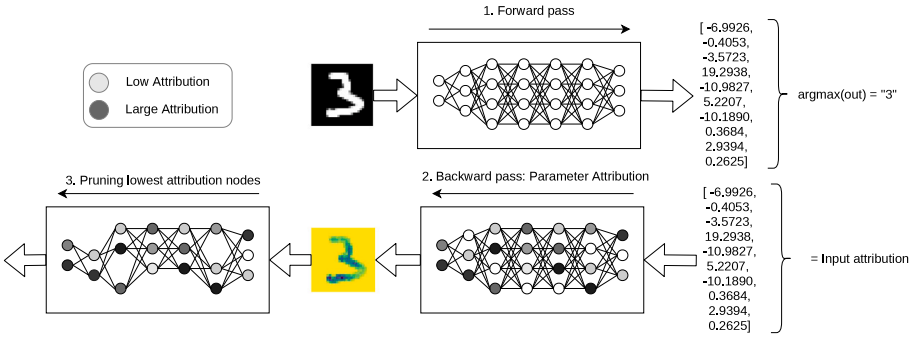


Fig. 1. Schematic overview of the pruning strategy for network specialization. Step 1: Forward pass through the network. Step 2: Use the output of the forward pass to calculate the attribution scores for the entire network using a backward pass. Step 3: Remove the most irrelevant neurons from the network.

In order to achieve the best results, the samples in the dataset should be evenly distributed over the target task. It is important to note that the target task can be a subset task, exactly the same task as the original, or a very related task compared to the original task.

This makes this method useful in two pruning use-cases:

1. **Pruning normally, preserving the original task**. Using original samples from the entire original domain, the network can be optimized without loss of knowledge of the original task. A retraining step may be needed to ensure the required performance is preserved.
2. **Pruning for specialization, preserving a subset of the original task**. Using only samples of the subset domain of the original task, or a related task to the original, the pruning method will specialize the network to focus only on this subset. This way, the network can be optimized with a loss of knowledge outside of the target task. This case is useful in a transfer learning setting where a pre-trained network will be used for a related problem. Furthermore, we expect even smaller footprints for pruned networks as more parameters of unnecessary concepts can be removed.

After the backpropagation step, we have attribution scores for each neuron/filter in the network. In our implementation, we propose two methods for combining the attribution scores of multiple samples. The first is by simply averaging the scores. This method will favor nodes with generally higher attribution scores over all samples, allowing it to better preserve all knowledge concepts in the network. The second method is taking the maximum attribution score for each node. This method will include all highest-scoring nodes, which works best for the specialization use-case as it allows for the dominant concepts of the target task to be better preserved. These scores are then used in the last step in order to remove the least contributing parameters in a structured manner.

5. Experiments

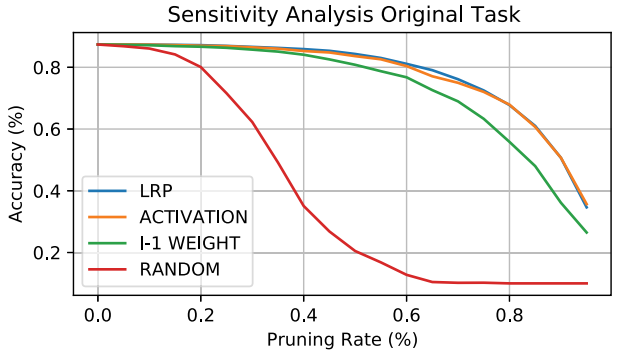
In order to evaluate which attribution method works best for each use-case, we perform several different experiments. The goal is to show comparative results of different attribution methods. The ultimate goal is to find the optimal method to minimize the computational cost of a model (number of parameters and FLOPS) while maximizing its performance on the target task (accuracy). In each experiment, we compare the two use-cases: maintaining the originally trained task of the model and the task specialization use-case. We use networks from the VGG and ResNet families, as well as LeNet-5 for demonstration and development purposes.

5.1. Layer-wise sensitivity analysis

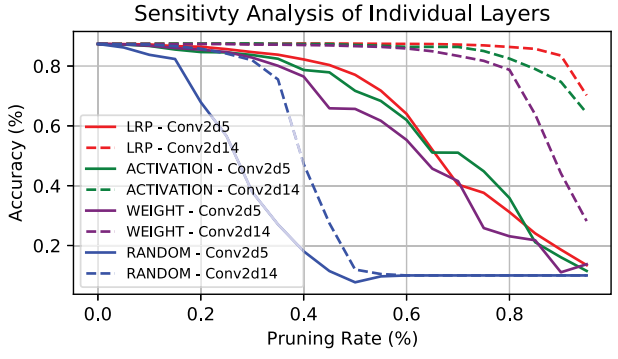
In the first experiment, we perform a layer-wise analysis of a network. We check how 'sensitive' each layer is to pruning. We measure the network's test accuracy after each pruning iteration with an increasing pruning rate. We compare the previously listed pruning criteria. It is important to note that the LRP-pruning criterion is the only method specifically designed to support specializing a network for a specific subset. We include the l_1-norm weight magnitude pruning criterion to compare what conventional pruning methods are capable of when only considering a subset of an original task. The random pruning criterion is included to show that the other pruning criteria can make intelligent decisions about which parts of the network to keep. This experiment aims to demonstrate the impact of pruning on a network's performance. We differentiate between two use-cases: the first is pruning while maintaining the originally trained task; the second is the specialization use-case.

We demonstrate this using an adapted version VGG-16 [33] network. The network is pre-trained on CIFAR10 [34], containing ten different classes. We pre-trained the network using Adam [35] for 100 epochs. We used PyTorch [36] for the implementation. The network is adapted to better fit the CIFAR10 problem.

The results are visualized in the plots shown in Figs. 2 and 3. Here the Test Accuracy (%) of the entire network is shown given a certain Pruning Rate (%) in a specific layer. In Figs. 2(a) and 3(a) we show the average test accuracy of all layers in the network combined. Figs. 2(b) and 3(b) show the test accuracy of two selected layers of the network separately.



(a)



(b)

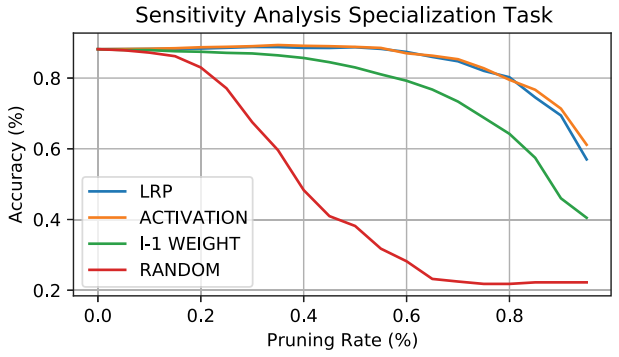
Fig. 2. Sensitivity analysis for VGG-16. These graphs show the (average) test accuracy of the network when pruning using a certain pruning rate in one layer. Results for the first use-case where we keep the original trained domain. (a) Average test accuracy of all layers. (b) Test accuracy when pruning one layer. Results are shown for convolutional layer 5 (Conv2d5) and layer 14 (Conv2d14).

When comparing Figs. 2(a) and 3(a), we can see that pruning for specialization is beneficial for the achieved pruning rate. The test accuracy remains greater for higher pruning rates. This follows our expectation that specializing the network for a specific subtask allows for certain encoded knowledge concepts to be removed from the network without impacting the test accuracy. In the first use-case, this is not the case, though, as the entire original task should be preserved. It is important to note that the size of the subset task for specialization does matter. In this experiment, the subtask consists of 3 classes. We chose this subtask as it shows a potentially realistic use-case of creating a specialized animal classifier from a more general model. The animal classes we chose are bird, cat, and dog. In Section 5.4 we discuss the impact of the target task size. Generally, these figures show that pruning using the LRP and activation criteria are more capable of specialization as well as maintaining the original task. The l_1-weight pruning method is not capable of this.

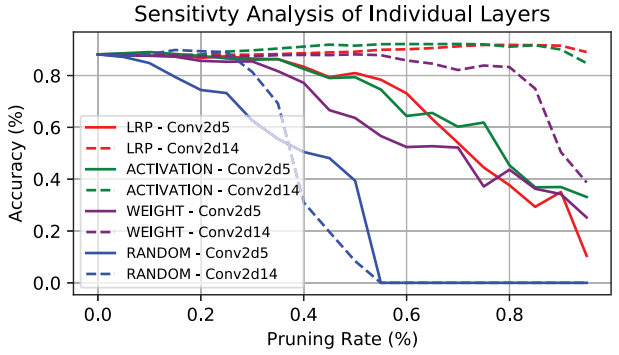
Finally, in Figs. 2(b) and 3(b), we see that layers towards the end of the network (which are larger) are less 'sensitive' to pruning. This can be explained as these layers encode higher-level concepts that are more specialized and more easily separated. The first layers of the network are more 'sensitive' as they encode much lower-level concepts which the rest of the network relies on. We can use this finding to guide where to prune to have the least amount of impact on the network's target task performance. Often it is more beneficial to prune more in the deeper layers as they tend to be designed much wider.

5.2. Overall compression results with retraining

In this section, we discuss the overall compression results of the layerwise-structured, iterative pruning method from Section 4 using different attribution methods. We compare different network architectures, more specifically, we use LeNet-5 [37], VGG-16 [33], and ResNet-50 [38]. LeNet-5 and VGG-16 are commonly used networks to demonstrate compression results. We include ResNet-50 as well to show results on a more modern network architecture. The results can be found in Table 1. In these experiments, we also differentiate between the two aforementioned use-cases. An important note is that these results were gathered with fine-tuning of the network after each iteration. This means even the random pruning criterion is able to maintain high accuracy. We generally see that specializing the network allows for smaller networks while maintaining good performance on the target task. Generally, all pruning criteria are able to maintain good performance. However, for the network specialization use-case, the LRP criterion is able to exceed the other criteria consistently in reduced resource cost while maintaining acceptable performance.



(a)



(b)

Fig. 3. Sensitivity analysis for VGG-16. These graphs show the (average) test accuracy of the network when pruning using a certain pruning rate in one layer. Results for the second use-case where we specialize the network to focus on a subset of the originally trained task. In this experiment, the subset task consists of the classes [bird, cat, dog]. These graphs show the higher possible pruning rates compared to the first use-case shown in Fig. 2. (a) Average test accuracy of all layers. (b) Test accuracy when pruning one layer. Results are shown for convolutional layer 5 (Conv2d5) and layer 14 (Conv2d14).

Table 1
Overall compression results on different network architectures using different pruning criteria. LeNet-5 is specialized on subset classes [1, 4, 8]. VGG-16 and ResNet-50 are specialized on subset classes [bird,cat,dog].

Model	Method	Original task			Specialized task		
		Test Acc. (%)	#Params.	#FLOPS	Test Acc. (%)	#Params.	#FLOPS
LeNet-5 (MNIST)	–	99.19	44,426	286,632	99.23	44,426	286,632
	l-1 weight	95.43	−83.23%	−44.90%	97.23	−90.31%	−65.23%
	Activation	96.17	−69.21%	−38.38%	95.78	**−95.66%**	−71.90%
	LRP	93.77	**−86.57%**	**−58.90%**	93.87	−94.79%	**−72.03%**
	Random	97.05	−68.50%	−23.93%	98.54	−71.66%	−49.28%
VGG-16 (CIFAR10)	–	94.35	33,646,666	332,942,336	89.60	33,646,666	332,942,336
	l-1 weight	91.58	−91.54%	−42.48%	90.40	−93.84%	−54.23%
	Activation	92.19	−85.16%	−38.30%	89.53	−95.91%	−53.69%
	LRP	92.03	**−93.61%**	**−48.84%**	88.40	**−96.51%**	**−58.81%**
	Random	92.21	−86.36%	−33.34%	82.70	−84.21%	−39.13%
ResNet-50 (CIFAR10)	–	93.65	23,520,841	327,624,704	95.90	23,520,841	327,624,704
	l-1 weight	89.69	−81.25%	**−56.88%**	95.07	−84.81%	−60.99%
	Activation	88.84	−82.96%	−52.82%	95.63	−86.03%	−59.12%
	LRP	87.78	**−83.53%**	−55.34%	95.13	**−86.45%**	**−62.37%**
	Random	88.59	−82.96%	−56.01%	94.45	−85.42%	−58.01%

5.3. Compression results without retraining

In other research [39], it is shown that retraining with a lot of data, for example, using the entire original dataset, decreases the impact of the selected pruning criteria on the final performance. Our results from Section 5.2 follow this finding. The final accuracy of the networks are all quite similar despite the pruning criteria used.

Table 2
Overall compression without retraining (fine-tuning) results on different network architectures using different pruning criteria. The pruning criteria are: $||w||_1 = l_1$ weight, ACT = Activation, LRP = Layerwise Relevance Propagation, RDM = Random. These are the same criteria as in the experiment from Section 5.2. Results are shown for different levels of target sparsity (Pruning Rate) in each layer. For the specialization use-case, VGG-16 and ResNet-34 are specialized on subset classes [bird, cat, dog] to maintain consistency.

Model	Pr. Rate (%)	Original task test Acc. (%)					Specialized task test Acc. (%)				
		–	$\|\|w\|\|_1$	ACT	LRP	RDM	–	$\|\|w\|\|_1$	ACT	LRP	RDM
VGG-16 (CIFAR10)	30%	94.35	75.54	73.85	**76.10**	16.69	89.60	79.25	54.37	**87.46**	34.29
	50%	94.35	26.04	23.42	**28.51**	10.10	89.60	53.77	33.58	**57.91**	30.25
	70%	94.35	**19.22**	9.87	10.02	9.81	89.60	34.09	0.00	**37.62**	0.00
ResNet-34 (CIFAR10)	30%	93.26	**90.96**	89.91	86.15	76.56	89.54	**87.77**	78.7	86.30	75.55
	50%	93.26	**82.20**	68.36	76.26	12.81	89.54	84.42	57.30	**86.26**	40.91
	70%	93.26	20.85	9.97	**26.42**	10.20	89.54	43.92	33.05	**44.10**	33.45

Therefore, in this experiment, we prune without retraining in order to determine which pruning criteria can best preserve the accuracy. In this experiment, we fix the pruning rate and measure the final test accuracy of the network. As previously mentioned, the first layers of a network encode very low-level concepts which are generic, and most outputs will rely on them. Therefore, pruning without retraining will have a considerable impact on the network, and thus we will not be removing parameters from the first four layers of the network.

Finally, we also repeat the same specialization and no specialization use-cases. In this experiment, the VGG-16 and ResNet-34 network architectures are used. We run the experiment with a given target sparsity level (pruning rate) in each layer. The results can be found in Table 2. Furthermore, these results show that pruning for task specialization allows for smaller networks while maintaining good performance. This follows our hypothesis from earlier that when specializing, certain encoded (learned) concepts can be safely removed from the network without impacting the target task performance of the network. From these results, we can also conclude that the LRP-based criterion is best suited for this task. This follows our result from the previous experiment as well, where retraining was allowed.

Finally, we can see the activation and random pruning criteria achieve 0.00% on the VGG-16 specialization experiment. This happened because both these criteria removed critical parts of the network architecture, making it no longer valid.

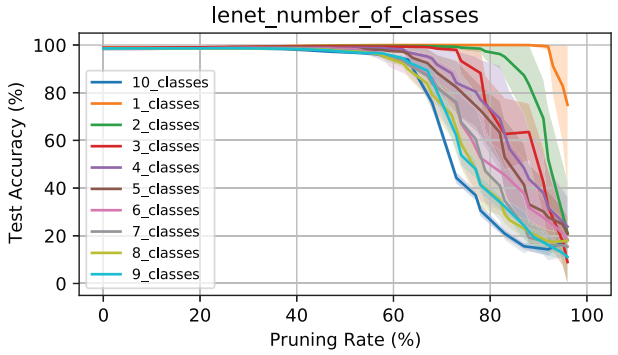
5.4. Target task size

Our previous experiments show the impact of specialization when adapting the network to a subset of size 3. However, the size of the subset will impact the compression performance. As the size of the target task gets bigger, the more originally trained concepts need to be preserved; hence the compression will be less. In this experiment, we change the subset and evaluate its impact on the compression performance. The results are shown in Fig. 4. When pruning for a small target task, more concepts (encoded by certain parameters) can be removed without impacting the performance. For this experiment, we pruned the networks using the LRP-based criterion as we concluded it performs best for network specialization. No fine-tuning was performed for this experiment.

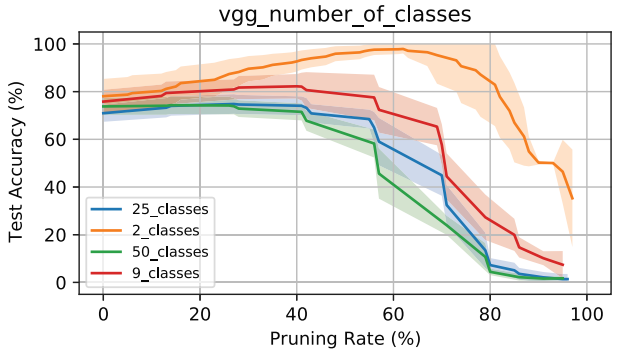
In the graphs in Fig. 4, we show the impact on both LeNet-5 and VGG-16. These graphs show the impact on two different tasks. The LeNet-5 trained network is trained on MNIST [40]. The VGG-16 network in this experiment was trained on the ILSVRC2012 dataset [41]. This shows the impact of network specialization pruning in large-scale applications such as ILSVRC2012. This dataset contains 1000 different classes. All these classes may not always be needed in a user's application. Therefore, network specialization can be very beneficial. From both graphs, we can conclude higher pruning rates are possible for smaller target tasks. This follows from the fact that fewer concepts have to be preserved. This aligns with our envisioned goal of specializing a network for a small subset of the originally trained task.

5.5. Impact on resources

Finally, we discuss our network specialization's impact on resource consumption. In order to verify the actual performance gain of the compressed networks, we performed experiments on two different embedded platforms. The first platform is the Nvidia Jetson TX2 [42]. This platform has an ARM processor in combination with a dedicated Nvidia GPU. The second platform is a Raspberry Pi Model 4B [43]. In the experiment, we compare the uncompressed VGG-16 and ResNet-50 networks with their LRP unspecialized and specialized compressed networks. We use LRP as we concluded this to be the most optimal attribution metric for network specialization in our previous experiments in Sections 5.1 and 5.2. The results are shown in Table 3. We show the run-time memory in megabytes (MB), the latency in milliseconds (ms), the number of parameters of the network, and the number of FLOPs of the network. From this table, we can conclude the improvements are substantial. The latency and run-time memory for both platforms are dramatically decreased. This shows the benefit of compressing neural networks for inference, especially for the specialization use-case where the total resource consumption is even lower. For VGG-16, the compressed specialized version is twice as fast on the GPU (Jetson) and almost 10 times faster on the CPU (Raspberry Pi). For ResNet-50, we see a speedup of 1.15 on the GPU and 2.3 on the CPU.



(a)



(b)

Fig. 4. Total network test accuracy as a function of the pruning rate when comparing different target task sizes for a LeNet-5 and VGG-16 network. The target tasks consist of randomly selected classes. The tasks are subsets of (a) MNIST [40] and (b) ILSVRC2012 [41]. The graphs show the mean test accuracy (line) and standard deviation (area) of 10 runs.

Table 3
Resource consumption measurements on Nvidia Jetson TX2 and Raspberry Pi 4 Model B.

Model	Uncompressed		Compressed original task		Compressed subtask	
	VGG-16	ResNet-50	VGG-16	ResNet-50	VGG-16	ResNet-50
#MParams	33.64	21.28	2.15	4.63	1.17	3.20
#MFLOPs	332.94	327.62	170.32	151.1	137.15	123.29
Nvidia Jetson TX2						
GPU Memory (MB)	513.21	89.97	18.28	17.92	**14.75**	**12.41**
Latency (ms)	26.30	52.77	13.12	46.75	**11.76**	**45.29**
Raspberry Pi 4 Model B						
CPU Memory (MB)	658.89	243.29	159.91	167.68	**152.92**	**158.25**
Latency (ms)	480.77	228.83	60.42	112.36	**49.33**	**99.80**

6. Conclusion and future work

This work evaluates the feasibility of neural network specialization using neural network compression. The overall goal is to achieve as small as possible network while maintaining high accuracy for a target task. We concentrate on a transfer learning use-case where we specialize a pre-trained neural network to focus only on a subtask of its originally trained task. This allows the network to only preserve the needed knowledge concepts of the originally trained task for the specialization task. This results in a less computationally demanding network compared to traditional pruning, where the original task is preserved.

We compare and demonstrate the capabilities of different pruning criteria using an iterative, layerwise-structured pruning algorithm. More specifically, we compare an XAI-inspired pruning criterion (LRP-based) to more traditional pruning criteria (weight-based, activation-based, and random). From our results, we can conclude that pruning for the specialization use-case is beneficial for the final computational cost of a network. The final computational cost across networks, when pruned for specialization, is also consistently lower. We showed this both with and without fine-tuning after pruning. Combining these results leads us to conclude

that the LRP-based pruning criterion is best-suited for this specialization used case. It can achieve smaller networks with fewer FLOPs while maintaining acceptable performance on the target task. The last experiment shows the impact of specialization pruning when executing the optimized models on embedded platforms. We showed that the optimized models use substantially fewer resources, allowing them to be more feasibly used for edge computing.

In future work, we want to extend the compression even further. In literature, the combination of neural network pruning and quantization is very common. For this reason, we would like to explore this for the network specialization use-cases as well to even further compress and accelerate the networks. Additionally, we will explore different methods of knowledge localization. Currently, using LRP, we localize knowledge concepts at a high level (outputs). We would like to extend this to lower-level semantic concepts to allow for even more flexibility in specializing the networks. Finally, we would like to explore more network architectures. In this work, we focused on networks trained for object classification. We would want to explore other tasks like detection, segmentation, etc.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research received funding from the Flemish Government (AI Research Program) and was supported by the Research Foundation Flanders (FWO) under Grant Number 1SA8122N.

References

[1] J. Redmon, S. Divvala, R. Girshick, A. Farhadi, You only look once: Unified, real-time object detection, in: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2016-December, 2016, pp. 779–788, http://dx.doi.org/10.1109/CVPR.2016.91, arXiv:1506.02640.
[2] D. Xu, D. Anguelov, A. Jain, PointFusion: Deep sensor fusion for 3D bounding box estimation, in: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, IEEE, 2018, pp. 244–253, http://dx.doi.org/10.1109/CVPR.2018.00033, http://arxiv.org/abs/1711.10871 https://ieeexplore.ieee.org/document/8578131/.
[3] A. Tao, K. Sapra, B. Catanzaro, Hierarchical multi-scale attention for semantic segmentation, (ISSN: 23318422) 2020, pp. 1–11, arXiv arXiv:2005.10821.
[4] T.B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D.M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, D. Amodei, Language models are few-shot learners, (ISSN: 23318422) 2020, arXiv arXiv:2005.14165.
[5] A. Baevski, H. Zhou, A. Mohamed, M. Auli, Wav2vec 2.0: A framework for self-supervised learning of speech representations, (ISSN: 23318422) 2020, pp. 1–12, arXiv, Figure 1 arXiv:2006.11477.
[6] K. Ullrich, E. Meeds, M. Welling, Soft weight-sharing for neural network compression, (ISSN: 23318422) 2017, pp. 1–16, arXiv arXiv:1702.04008.
[7] A. Boulch, ShaResNet: reducing residual network parameter number by sharing weights, (ISSN: 23318422) 2017, arXiv arXiv:1702.08782.
[8] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, D. Kalenichenko, Quantization and training of neural networks for efficient integer-arithmetic-only inference, Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recogn. (2018) 2704–2713, http://dx.doi.org/10.1109/CVPR.2018.00286, arXiv:1712.05877.
[9] G. Hinton, O. Vinyals, J. Dean, Distilling the knowledge in a neural network, 2015, pp. 1–9, arXiv:1503.02531.
[10] J. Frankle, M. Carbin, The lottery ticket hypothesis: Finding sparse, trainable neural networks, 2018, pp. 1–42, arXiv:1803.03635.
[11] M. Hagiwara, Removal of hidden units and weights for back propagation networks, in: Proceedings of 1993 International Conference on Neural Networks (IJCNN-93-Nagoya, Japan) vol.1, 1993, pp. 351–354, http://dx.doi.org/10.1109/IJCNN.1993.713929.
[12] Y. LeCun, J.S. Denker, S.A. Solla, Optimal brain damage (pruning), Adv. Neural Inf. Process. Syst. (1990) 598–605.
[13] S. Han, H. Mao, W.J. Dally, Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, 2015, pp. 1–14, arXiv:1510.00149.
[14] S. Anwar, K. Hwang, W. Sung, Structured pruning of deep convolutional neural networks, 21 (4), (ISSN: 0147-5185) 2015, pp. 399–406, http://dx.doi.org/10.1097/00000478-199704000-00005, arXiv:1512.08571.
[15] G. Georgiadis, Accelerating convolutional neural networks via activation map compression, Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recogn. 2019-June (2018) 7078–7088, http://dx.doi.org/10.1109/CVPR.2019.00725, arXiv:1812.04056.
[16] X. Sun, X. Ren, S. Ma, H. Wang, Meprop: Sparsified back propagation for accelerated deep learning with reduced overfitting, in: 34th International Conference on Machine Learning, vol. 7, ICML 2017, ISBN: 9781510855144, 2017, pp. 5080–5089, arXiv:1706.06197.
[17] C. Liu, H. Wu, Channel pruning based on mean gradient for accelerating convolutional neural networks, Signal Process. 156 (2019) 84–91, http://dx.doi.org/10.1016/j.sigpro.2018.10.019, URL https://linkinghub.elsevier.com/retrieve/pii/S0165168418303517.
[18] X. Dong, S. Chen, S.J. Pan, Learning to prune deep neural networks via layer-wise optimal brain surgeon, Adv. Neural Inf. Process. Syst. 2017-Decem (Nips) (2017) 4858–4868, arXiv:1705.07565.
[19] N. Lee, T. Ajanthan, P.H.S. Torr, SNIP: single-shot network pruning based on connection sensitivity, 2018, CoRR abs/1810.02340 arXiv:1810.02340.
[20] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, W. Samek, On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation, in: O.D. Suarez (Ed.), PLoS One 10 (7) (2015) e0130140, http://dx.doi.org/10.1371/journal.pone.0130140.
[21] S.-K. Yeom, P. Seegerer, S. Lapuschkin, A. Binder, S. Wiedemann, K.-R. Müller, W. Samek, Pruning by explaining: A novel criterion for deep neural network pruning, Pattern Recognit. 115 (2021) 107899, http://dx.doi.org/10.1016/j.patcog.2021.107899, arXiv:1912.08881.
[22] D. Balemans, W. Casteels, S. Vanneste, J. de Hoog, S. Mercelis, P. Hellinckx, Resource efficient sensor fusion by knowledge-based network pruning, Internet of Things 11 (2020) 100231, http://dx.doi.org/10.1016/j.iot.2020.100231.
[23] P. Molchanov, S. Tyree, T. Karras, T. Aila, J. Kautz, Pruning convolutional neural networks for resource efficient inference, in: 5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings, (2015) 2017, pp. 1–17, arXiv:1611.06440.
[24] N. Lee, T. Ajanthan, P.H.S. Torr, SNIP: Single-shot network pruning based on connection sensitivity, in: 7th International Conference on Learning Representations, ICLR 2019, 2018, arXiv:1810.02340.
[25] B. Liu, Y. Cai, Y. Guo, X. Chen, TransTailor: Pruning the pre-trained model for improved transfer learning, 2021, http://dx.doi.org/10.48550/arXiv.2103.01542, URL http://arxiv.org/abs/2103.01542.

[26] B.W. Mateusz Żarski, K. Książek, J.A. Miszczak, Finicky transfer learning—A method of pruning convolutional neural networks for cracks classification on edge devices, Comput.-Aided Civ. Infrastruct. Eng. 37 (2021) 500–515, http://dx.doi.org/10.1111/mice.12755.
[27] A.G. Tejalal Choudhary, J. Sarangapani, A transfer learning with structured filter pruning approach for improved breast cancer classification on point-of-care devices, Comput. Biol. Med. 134 (2021) http://dx.doi.org/10.1016/j.compbiomed.2021.104432.
[28] C. Alippi, S. Disabato, M. Roveri, Moving convolutional neural networks to embedded systems: The AlexNet and VGG-16 case, in: 2018 17th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN), IEEE, 2018, pp. 212–223, http://dx.doi.org/10.1109/IPSN.2018.00049, URL https://ieeexplore.ieee.org/document/8480072/.
[29] H. Li, A. Kadav, I. Durdanovic, H. Samet, H.P. Graf, Pruning filters for efficient ConvNets, in: 5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings, (2016) 2016, pp. 1–13, arXiv:1608.08710.
[30] A. Ardakani, C. Condo, W.J. Gross, Activation pruning of deep convolutional neural networks, in: 2017 IEEE Global Conference on Signal and Information Processing, GlobalSIP, 2017, pp. 1325–1329, http://dx.doi.org/10.1109/GlobalSIP.2017.8309176.
[31] S. Han, J. Pool, J. Tran, W.J. Dally, Learning both weights and connections for efficient neural networks, Adv. Neural Inf. Process. Syst. 2015-Janua (2015) 1135–1143, arXiv:1506.02626.
[32] E.J. Crowley, J. Turner, A. Storkey, M. O'Boyle, A closer look at structured pruning for neural network compression, 2018, pp. 1–12, arXiv:1810.04622.
[33] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, in: 3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings, 2014, arXiv:1409.1556.
[34] A. Krizhevsky, Learning multiple layers of features from tiny images, tech. rep., 2009.
[35] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2014, http://dx.doi.org/10.48550/arXiv.1412.6980, URL http://arxiv.org/abs/1412.6980.
[36] Pytorch, 2021, https://pytorch.org/. (Accessed 08 November 2021).
[37] Y. LeCun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, L.D. Jackel, Backpropagation applied to handwritten zip code recognition, Neural Comput. 1 (4) (1989) 541–551, http://dx.doi.org/10.1162/neco.1989.1.4.541.
[38] K. He, X. Zhang, S. Ren, J. Sun, Deep Residual Learning for Image Recognition, arXiv:1512.03385v1.
[39] M.M.K. Deepak Mittal, B. Ravindran, Studying the plasticity in deep convolutional neural networks using random pruning, Mach. Vis. Appl. 30 (2019) 203–216, http://dx.doi.org/10.1007/s00138-018-01001-9.
[40] Y. Lecun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86 (11) (1998) 2278–2324, http://dx.doi.org/10.1109/5.726791.
[41] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A.C. Berg, L. Fei-Fei, ImageNet large scale visual recognition challenge, Int. J. Comput. Vis. (IJCV) 115 (3) (2015) 211–252, http://dx.doi.org/10.1007/s11263-015-0816-y.
[42] Nvidia jetson TX2 developer, 2021, https://developer.nvidia.com/EMBEDDED/jetson-tx2. (Accessed 10 November 2021).
[43] Raspberry Pi 4 model B specifications – raspberry Pi, 2021, https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/. (Accessed 10 November 2021).