DNN pruning with principal component analysis and connection importance estimation☆

Marc Riera *, José María Arnau, Antonio González

Universitat Politècnica de Catalunya (UPC), Barcelona, Spain

ARTICLE INFO

ABSTRACT

DNN pruning reduces memory footprint and computational work of DNN-based solutions to improve performance and energy-efficiency. An effective pruning scheme should be able to systematically remove connections and/or neurons that are unnecessary or redundant, reducing the DNN size without any loss in accuracy. In this paper we show that some of the most popular pruning schemes, such as the Near Zero Weights, require an extremely time-consuming iterative process that requires retraining the DNN many times to tune the pruning hyperparameters. Then, we propose a DNN pruning scheme based on Principal Component Analysis and relative importance of each neuron's connection (PCA+DIRIE) that automatically finds the optimized DNN in one shot without requiring hand-tuning of multiple parameters. The experimental results show the effectiveness of our method on several benchmarks. Notably, on ImageNet, PCA+DIRIE can prune up to 60% of ResNet-50 with negligible impact on accuracy.

1. Introduction

DNN pruning has attracted the attention of the research community in recent years [1–4]. Based on the observation that DNN models tend to be oversized and include a high degree of redundancy, pruning aims at reducing the model size by removing unimportant connections and/or neurons. The pruned model retains accuracy while requiring significantly less memory storage and computations, resulting in large performance improvements and energy savings.

Pruning requires the pruned model to be retrained; otherwise, the effectiveness of pruning is dramatically reduced. Finding the appropriate amount of pruning on each layer is a key factor that determines the efficiency of the scheme. If the pruning is too aggressive the DNN will not recover its accuracy after retraining, whereas if the pruning is too conservative, an opportunity to further optimize the DNN is lost. Most of the previously proposed DNN pruning schemes set the amount of pruning based on an expensive design space exploration or sensitivity analysis [1]. We argue that these approaches are impractical for very deep neural networks, as they require retraining the DNN a large number of times, and each one may take several hours or days even on a high-end GPU for each retraining. To exacerbate the problem, hyperparameters such as learning rate or weight decay have to be manually tuned [1,3], further increasing the search space.

In this paper, we present a novel DNN pruning scheme that does not require such an expensive search to find the percentage of pruning

for each layer, and it does not require to tune any hyperparameter. We refer to it as PCA plus Distribution Invariant Relative Importance Estimation (PCA+DIRIE) pruning. Dimension reduction methods such as PCA and SVD have recently become very popular for their effectiveness in DNN pruning [5,6]. Our scheme first applies node (i.e. neuron) pruning through PCA. We consider each layer as a system that produces "information" encoded by an N-dimensional array, where N is the number of neurons. The goal of pruning is to reduce the number of neurons to M (M < N) without loosing "information". The optimal pruning should find the minimum M for each level.

Principal Component Analysis (PCA) is a well-known statistical procedure that transforms a set of N-dimensional variables to a new coordinate system in which all coordinates are orthogonal and ordered from highest to lowest variance. Since variance can be considered as a good proxy of amount of "information", our scheme exploits PCA to determine the amount of neurons that can be safely removed. Although this is not the first work that proposes PCA for neuron pruning, we show that prior schemes [7,8] are ineffective for modern DNNs in Section 2, and propose a different mechanism to effectively apply PCA-based node pruning in Section 4.1.

Once the percentage of pruning for a layer is set, our scheme has to select which neurons are removed. We have evaluated prior heuristics for node pruning [9,10] and found that, if retraining is applied, they achieve the same results as a blind node pruning that randomly selects

the nodes to be removed from the model. Our experimental results show that the only relevant parameter is the amount of pruning, i.e. percentage of nodes to be pruned, and not which specific nodes are actually removed, since the topology of the DNN is not affected by that decision and the retraining will adjust the weights of the non-pruned nodes. Note that some previous works [11–13] reached similar conclusions with a different set of applications and DNNs.

Once we determine the minimum number of neurons for each layer, there are still further opportunities to prune at the connection level. The first step, i.e. node pruning, results in layers that are not sparse since full neurons with all their connections are removed or kept. However, for a given non-pruned neuron there may be connections that are unimportant. After the PCA-based node pruning, our scheme estimates the relative importance of each connection with respect to the other connections of the same neuron. Those connections with a low importance are removed and the final network is retrained. Unlike the node pruning step where the heuristic to select neurons is irrelevant, our results show that the heuristic used to choose the connections to be pruned has a non-negligible impact, achieving an additional 10%–30% of pruning over randomly choosing the connections. The overall scheme is non-iterative: it consists of only two steps and requires a single retraining after each of these steps. We show that this scheme produces results similar to or better than previously proposed iterative approaches that require an expensive (unfeasible for large networks) search.

To summarize, this paper focuses on DNN pruning methodologies. We highlight the weaknesses of some popular pruning methods and solve them by proposing a more effective and practical scheme. The main contributions of this paper are the following:

- We analyze a selection of popular pruning methods and make two key observations.

    - First, for node pruning the heuristics used to decide what neurons to prune are irrelevant; the only important parameter is how much to prune, since the retraining adjusts the weights of the remaining neurons. On the other hand, the heuristic used for pruning connections is relevant as it may increase the amount of pruning by 10%–30% over a random selection.
    - Second, most of the previous popular pruning methods require an unrealistic number of parameters to be manually configured by trial and error for each DNN to be pruned.

- We propose a novel, two-step pruning method that overcomes the above weaknesses. First, redundant nodes are removed by performing a PCA analysis of the outcome of each layer. Second, remaining unimportant connections are removed by taking into account their contribution relative to the rest of connections for a given neuron. PCA+DIRIE provides a pruning of 65% on average after evaluating it for multiple DNNs, and it is not iterative.

The rest of the paper is organized as follows. Section 2 reviews some popular pruning methods. Section 3 presents an analysis of these pruning methods and a discussion about their effectiveness and practicality. Section 4 presents a new pruning method based on PCA and relative connection's importance. Section 5 describes the evaluation methodology. Section 6 discusses the experimental results. Section 7 reviews the related work. Finally, Section 8 sums up the main conclusions of this work.

2. Main DNN pruning schemes

A variety of different DNN pruning schemes have been recently proposed. Pruning reduces both the model size and the number of computations with the aim of reducing energy consumption and increasing performance. In general, pruning schemes require retraining of

the pruned network to recover the original accuracy, and the resulting model becomes sparse which may incur in some overheads depending on the system, since working with sparse matrices/vectors is more costly than operating on dense arrays for many current systems.

Although pruning methods have achieved tremendous success for image classification (e.g. AlexNet [14,15]), there are very few studies about their effectiveness for other applications such as speech recognition. More importantly, most of the schemes rely on heuristics with multiple parameters that require manual tuning, including the percentage of pruning for each layer. The next subsections provide more details on some of the most recent and popular pruning methods.

2.1. Near zero weights pruning

Han et al. [1] proposed a pruning method to remove the connections whose weight has an absolute value lower than a given threshold, which is computed using the following equation:

$$Threshold = std(W_l) * qp \qquad (1)$$

where $std(W_l)$ represents the standard deviation of all weights in layer l and the quality parameter (qp) determines the degree of pruning. The main idea of this heuristic is to remove the weights that are closer to zero. In the paper they report a 90% pruning for AlexNet without accuracy loss. However, the quality parameter is different per layer and the paper does not present any methodology to set it up other than try and error. Note that exploring all possible combinations would be totally unfeasible for networks with many layers since each trial must be followed by a retraining, which is extremely expensive. Even for AlexNet, which has only 8 layers, the exploration of the design space is huge. For more recent DNNs such as ResNet [16], the winner of the 2015 Imagenet Large Scale Visual Recognition Challenge [17], DenseNet [18] or SENet [19], the winner for 2017, this would be impractical given that they have more than 100 layers.

We implemented this method using a global quality parameter for all the layers, to reduce the search space to just one parameter, and the degree of pruning achieved was more moderate, around 70% in AlexNet. Finally, note that the heuristic they used does not work well if weights are not distributed around zero. In this case, this heuristic does not remove any connection, but there still may be connections that are unimportant compared to the others. For instance, if a neuron has ten input connections, nine of them have weights with a magnitude around 100 and the remaining one has a magnitude of around 10, this latter connection will likely be unimportant. However, this heuristic will not remove it since its weight is much larger than zero.

2.2. Node pruning

He et al. [9] proposed multiple importance metrics to identify which nodes i are redundant for each layer l of a DNN. After training the DNN, a score is calculated for each node $S(i, l)$ using one of the metrics. These importance metrics are defined as below.

- Entropy:

$$Score(i, l) = \frac{d_i^l(O)}{|O|} * \log_2(\frac{d_i^l(O)}{|O|}) + \frac{a_i^l(O)}{|O|} * \log_2(\frac{a_i^l(O)}{|O|}) \qquad (2)$$

where $|O|$ is the total number of inputs (i.e. audio frames, images...) in dataset O (e.g. the whole training dataset), and $a_i^l(O)$ and $d_i^l(O)$ are the total number of inputs which activate or de-activate node i of layer l, respectively ($a_i^l(O) + d_i^l(O) = |O|$), after executing the DNN with the corresponding inputs. A node is activated if the output value is greater than a threshold.

- Output Weights Norm (o-norm):

$$Score(i, l) = \frac{1}{N^{l+1}} \sum_{j=1}^{N^{l+1}} \left| W_{ij}^{l+1} \right| \qquad (3)$$

where N^{l+1} is the number of nodes of the layer $l + 1$.

- Input Weights Norm (i-norm):

$$Score(i, l) = \frac{1}{N^{l-1}} \sum_{j=1}^{N^{l-1}} \left| W_{ji}^l \right| \qquad (4)$$

where N^{l-1} is the number of nodes of the layer $l - 1$.

The first metric, called Entropy, examines the activation distribution of each node. A node is considered activated if the output value is greater than a threshold of 0,5. The idea of this metric is that if one node's outputs are almost identical on all training data, these outputs do not generate variations to later layers and consequently they are not useful. The second and third metrics, called i/o-norm, determines the importance of a neuron based on the average of the weights of its incoming or outgoing connections. Nodes are sorted by their scores and those with lower scores are removed along all relevant incoming and outgoing connections. The network is then retrained.

All the metrics achieve similar results, around 60% of pruning on TIMIT, a DNN for speech recognition. Note that one still has to decide how much to prune each layer, and they do not provide any heuristic to determine this other than trial and error. Furthermore, we show in Section 3 that a blind pruning that randomly selects the nodes to be removed achieves the same results as the aforementioned heuristics.

2.3. Similarity pruning

Another way to detect redundancy, proposed by Srinivas et al. [10], is to measure how similar the nodes are by computing the squared difference of the weights for each pair of nodes (i, j) of layer l using the following equation:

$$Saliency(i, j, l) = \sum_{k=1}^{N} (\left\| W_{ik} - W_{jk} \right\|)^2 \qquad (5)$$

The neurons with the lowest saliency are pruned. In this scheme, retraining is not applied but they achieve a rather moderate 35% of pruning on AlexNet, which is low compared to other methods. This method is only applied to the fully-connected layers. In short, this method avoids retraining but it achieves a low percentage of pruning and requires a huge space exploration to determine the particular threshold that should be used for each network layer.

2.4. Scalpel

Scalpel [3] is an iterative pruning method that determines which nodes to prune during training. To this end, a mask node is added after each original neuron to multiply its output by a parameter alpha that can be either 1 or 0. The method is divided into two steps, in the first step the mask layers are trained depending on a weight decay parameter that determines how much aggressive the pruning is. Then, the nodes for which the mask becomes zero after the training are removed and the network is retrained without the masks layers. This process is repeated multiple times, each time with an increased value of the weight decay, until a loss in accuracy is observed.

For the training phase when the masks are added, Scalpel uses two sets of variables, called alphas and betas. Alphas represent the pruning mask that is applied to the output of the nodes y_i during the forward evaluation of the network. That is:

$$Y_i' = \alpha_i * y_i \qquad (6)$$

where Y_i' are the final outputs passed to the next layer.

Alphas cannot be learned by the conventional Backpropagation method [20], since the cost function is not a continuous function of alpha. To overcome this, Scalpel associates another parameter called beta to each alpha. These betas can take any Real value and are learned during training, as if they were the multiplicative coefficients applied to the outputs. In each training iteration k, alphas are updated to 0 or

1 depending on the betas, a threshold (T) and a epsilon offset as shown in Eq. (7).

$$\alpha_{i|k} = \begin{cases} 1 & T + \epsilon \le \beta_{i|k} \\ \alpha_{i|k-1} & T \le \beta_{i|k} < T + \epsilon \\ 0 & \beta_{i|k} < T \end{cases} \qquad (7)$$

Regularization is applied to the betas during training to penalize high values by using a weight decay parameter. The weight decay parameter also determines the amount of pruning since a high value will increase the penalization to the betas and the backpropagation algorithm will try to reduce them, and lower betas imply that more alphas are zero, which results in more pruning. Finally, multiple iterations of the algorithm are done by increasing the weight decay parameter on each iteration to increase the pruning, until a loss in accuracy is observed. The weight decay and the step that is used to increase it on each iteration have to be manually set and are different for each DNN.

The main drawback of Scalpel is that it requires manual tuning of multiple parameters (threshold, epsilon, learning rate, etc.) for each particular DNN. Besides, its pruning effectiveness is not better than using previous methods, since its main target is pruning the DNN while avoiding sparsity. For instance, Scalpel achieves only 20% of node pruning for AlexNet.

2.5. PCA pruning

Levin et al. [7] proposed a pruning method to remove the nodes by using Principal Components Analysis (PCA). PCA can be used to reduce the number of nodes by computing the correlation matrix of the nodes activity. We can perform an eigendecomposition of the correlation matrix of the nodes activity to obtain the eigenvectors and eigenvalues. The eigenvalues can be used to rank the importance of a node of the new system.

The algorithm they propose to prune is divided into multiple steps. Starting from the first layer, the correlation matrix of the nodes activity is computed. The nodes activity is measured from multiple inputs of the training set and a pretrained network. The eigenvectors of the correlation matrix (i.e. Principal Components) are ranked by their corresponding eigenvalue and the effect of removing each node (i.e. eigenvector) is measured using the validation set. The nodes that do not increase the error are chosen to be removed. The weights of the layer are projected into the new subspace by multiplying the original weights (W) by the significant eigenvectors (C_l) as shown in Eq. (8). The procedure continues until all the layers are pruned. Note that this algorithm follows an iterative pruning since the validation and projection is performed after removing each node and stops when the accuracy of the network decreases. This method is only applied to the fully-connected layers and does not require any additional retraining.

$$W -> W * C_l * C_l^T \qquad (8)$$

$$W * I -> W * C_l * C_l^T * I \qquad (9)$$

Note that this scheme does not actually prune physical nodes or connections but reduces the number of parameters and computations depending on the amount of principal components that are removed. However, since the inputs (I) also have to be projected using the significant eigenvectors as shown in Eq. (9), both the non-pruned eigenvectors and the projected weights have to be stored. Therefore, the pruning of eigenvectors has to be highly effective in order to actually reduce parameters and computations of the neural network.

The method was originally evaluated using an small feed-forward network of two layers with a time series dataset. We implemented this method for LeNet5 using the MNIST dataset and it achieved only around 10% of pruning with negligible accuracy loss. We have also tested this method on a modern Kaldi DNN [21] and the pruning achieved was less than 1%. These low pruning percentages compared to previous methods suggest that pruning effectiveness is quite limited if the pruning method does not include retraining.

Table 1

DNNs employed for the pruning comparison. Kaldi [22] is an MLP for acoustic scoring trained on Librispeech [23], AlexNet [15] is a CNN for image classification trained on ImageNet [17], and LeNet5 is a CNN for digit classification trained on MNIST [24]. The table only includes FC and CONV layers, as these layers take up the bulk of computations in DNNs. Other layers, such as ReLU or Pooling, are not shown for the sake of simplicity.

Kaldi (18MB)			AlexNet (200MB)				LeNet5 (12MB)			
Accuracy: 89.51%			Accuracy: 57.48%				Accuracy: 99.34%			
Layer	Input Dim	Output Dim	Layer	In Dim	Out Dim	Kernel	Layer	In Dim	Out Dim	Kernel
FC1	360	360	CONV1	3*224*224	64*55*55	11*11	CONV1	1*28*28	32*28*28	5*5
FC2	360	2000	CONV2	64*27*27	192*27*27	5*5	CONV2	32*14*14	64*14*14	5*5
FC3	400	2000	CONV3	192*13*13	384*13*13	3*3	FC1	7*7*64	1024	–
FC4	400	2000	CONV4	384*13*13	384*13*13	3*3	FC2	1024	10	–
FC5	400	2000	CONV5	384*13*13	256*13*13	3*3				
FC6	400	3482	FC1	5*5*256	4096	–				
			FC2	4096	4096	–				
			FC3	4096	1000	–				

3. Weaknesses of previous pruning schemes

In this section, we highlight the main weaknesses of popular DNN pruning schemes. For quantitative evaluations, we use three different DNNs, whose parameters are shown in Table 1. Kaldi is a Multi-Level Perceptron (MLP) for acoustic scoring, a key task of a speech recognition system. It takes as input a window of 9 frames of speech (current frame and the four previous and four next frames), where each frame is represented as an array of 40 features. Kaldi DNN generates the likelihoods of the 3482 senones, where a senone represents part of a phoneme. On the other hand, LeNet5 and AlexNet are popular Convolutional Neural Networks (CNNs). LeNet5 is a small CNN to classify written digits. Finally, AlexNet is a CNN for classifying color images into 1000 possible classes that range from different animals to various types of objects.

We first evaluate the effectiveness of the schemes previously proposed to select the connections and/or nodes that are removed from the model, and compared them with a blind pruning scheme that randomly selects the connections/nodes. We report the accuracy loss and the amount of pruning achieved for the networks shown in Table 1. More specifically, we implemented the near-zero pruning (which applies to connections), and two of the node pruning methods: the similarity and the i-norm pruning (see Section 2). Although similarity pruning does not use retraining, we include it in all the methods for a fair comparison. We report results for different overall pruning percentages starting from 10% and increasing it by steps of 10%. The parameters of each pruning scheme are manually adjusted to attain the target percentage of global pruning. For instance, for near-zero pruning, we tried different values of the *quality parameter (qp)* until the target percentage of pruning was attained. Note that the percentage of pruning applied to each individual layer is determined by the particular heuristics used by each method and is not uniform across layers, i.e. some layers are pruned more aggressively than others.

Fig. 1 shows the comparison between near-zero and random pruning of connections in terms of Word Error Rate (WER is the main metric used in speech recognition; lower is better) for Kaldi DNN. We can observe that for 10%–20% pruning both methods achieve very similar accuracy. For 30%–80% pruning, near-zero is slightly better, and for 90% pruning random is slightly more accurate. Random pruning achieves up to 50% of pruning with negligible accuracy loss, whereas near-zero pruning achieves up to 70%. Therefore, the near-zero scheme can achieve up to 20% more pruning, but the random scheme still performs quite well. To further reinforce this conclusion, we performed multiple tests with the random scheme using different seeds to obtain different pruning patterns. For all the random experiments the accuracy obtained after retraining was almost the same, with smaller differences of less than 0.2%.

Fig. 2 shows the comparison between the different methods to prune nodes for the Kaldi DNN. We observe very minor differences in terms of accuracy among all the methods, so there is no clear winner. For node pruning, the last layer cannot be pruned since these neurons generate
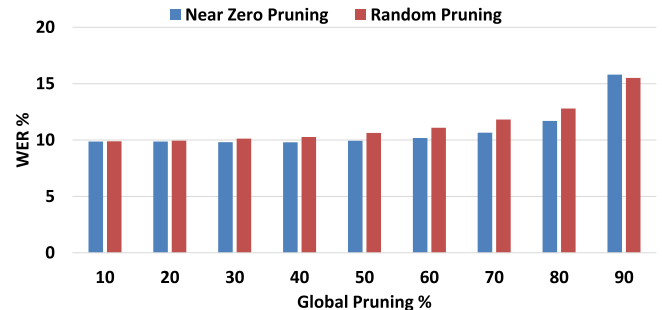

Fig. 1. Comparison between near-zero and random pruning of connections of Kaldi on Librispeech for different percentages of global pruning.
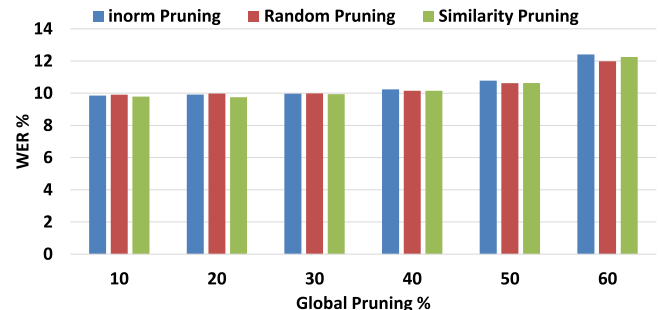

Fig. 2. Comparison between i-norm, similarity and random pruning of nodes of Kaldi on Librispeech for different percentages of global pruning.

the output values that are used by the application. For instance, in Kaldi these are the probabilities used by the Viterbi beam search. Therefore, the maximum degree of pruning that can be achieved is around 60% of the nodes. We can see that the differences in WER are less than 1% in all the cases, random being slightly better when the global pruning is high (50% and 60%).

Results for LeNet5 are shown in Fig. 3 for link pruning and Fig. 4 for node pruning. In this case, LeNet5 has a high tolerance to errors and the accuracy is well maintained until pruning 90% of the network for both types of pruning. At that point, the accuracy starts to decrease, being the random scheme slightly worse in the case of link pruning, but only by around 1%, whereas for node pruning there are no significant differences between random, i-norm and similarity schemes for all percentages of pruning.

Fig. 5 shows the comparison between near-zero and random pruning of connections in terms of Top-1 accuracy for AlexNet. We can observe that up to 50% of pruning the accuracy is largely recovered for both methods. Then, the random pruning starts to decrease the accuracy while the near-zero is able to maintain it until 80%, and beyond that point it drops significantly.
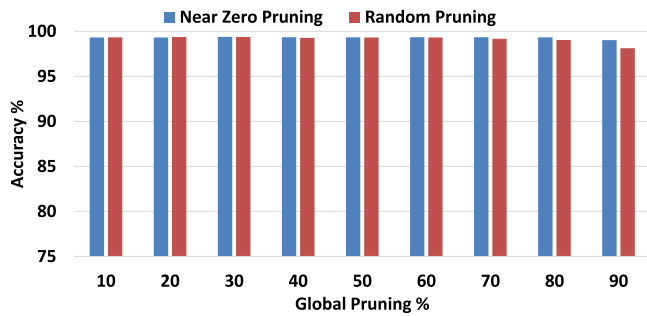
Fig. 3. Comparison between near-zero and random pruning of connections of LeNet5 on MNIST for different percentages of global pruning.
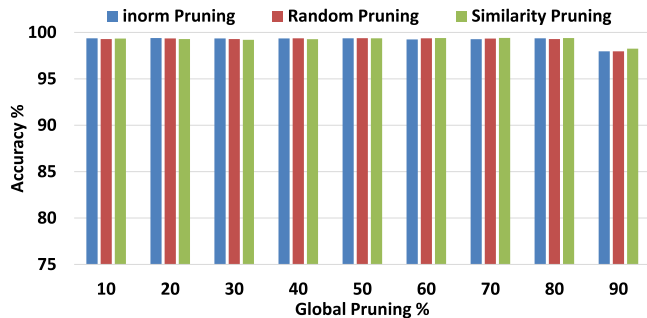

Fig. 4. Comparison between i-norm, similarity and random pruning of nodes of LeNet5 on MNIST for different percentages of global pruning.
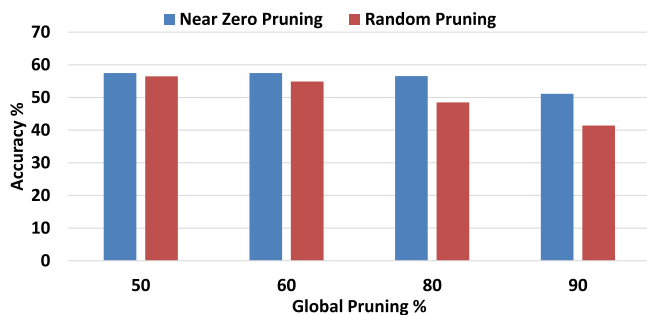

Fig. 5. Comparison between near-zero and random pruning of connections of AlexNet on ImageNet for different percentages of global pruning.

Note that some studies in the literature report results for pruning that require the exploration of a huge design space, which may be impractical for some DNNs. For instance, the near-zero pruning scheme could obtain better results by applying a different quality parameter (qp) for each network layer, as reported in the original paper [1]. However, trying different values of qp for each layer requires to explore an exponential number of configurations (exponential with the number of layers). Taking into account that for each configuration a retraining is needed, and retraining is extremely costly (can take several days in large networks), we claim that such strategies based on exploring an exponential number of configurations are impractical for many contemporary DNNs, which have hundreds of layers.

In summary, we observed that in terms of accuracy the random node pruning behaves almost equal to the rest of the methods. On the other hand, the random pruning of connections is somewhat less effective than the analyzed schemes. In other words, the heuristics to choose which neurons to prune are irrelevant while the heuristics to choose the connections may impact the final result. Previous works [11] have done a similar analysis with a different set of applications and neural

networks (i.e. VGG [25], ResNet [16] and DenseNet [18]) which confirms our observations. Besides, a main weakness of most of the popular pruning methods is their need of tuning one or multiple parameters through a trial and error process. This is extremely costly since each experiment requires retraining the network, and is impractical if the number of configurations to explore is too high (e.g. exponential with number of network layers).

4. PCA+DIRIE pruning method

Our proposed pruning method consists of two main steps. First, it performs a node pruning based on a PCA analysis of the data produced by each fully-connected layer. Next, some of the remaining connections are pruned based on their importance relative to the rest of incoming connections of the same neuron. The proposed scheme is not iterative and requires only two retraining operations, one after each of the two steps. We refer to it as PCA plus Distribution Invariant Relative Importance Estimation (PCA+DIRIE) pruning (https://github.com/upc-arco/PCA-DIRIE-Pruning).

4.1. Node pruning through PCA

The first step of the proposed approach is to prune redundant neurons in each layer through a Principal Components Analysis (PCA) [26]. PCA is a well-known statistical method to summarize data, and is typically used to reduce the dimensionality of a dataset. PCA transforms a set of observations from different variables with high correlation into a set of principal components without linear correlation. Therefore, one of its main usages is to determine redundancy.

In our context, PCA is used to reduce the number of nodes of a layer. Each layer can be regarded as a system that for each evaluation generates an output value represented as a n-dimensional vector, where n is the number of neurons of this layer. PCA allows us to represent a set of n-dimensional values in a different coordinate system, without loosing any information, by applying a linear transformation:

$$New_1 = Old_1 * a_1 + Old_2 * b_1 + \cdots + Old_n * z_1$$
$$New_2 = Old_1 * a_2 + Old_2 * b_2 + \cdots + Old_n * z_2$$
$$\cdots \tag{10}$$
$$New_n = Old_1 * a_n + Old_2 * b_n + \cdots + Old_n * z_n$$

Besides, in the new system the components are orthogonal (i.e., they do not contain any redundant information) and are ordered from higher to lower variance (i.e., from more to less information). If the original data presents high correlation among some of the n dimensions, in the transformed coordinate system, the last components will have very low variance. If we remove these low-variance components, we can represent the data in a lower-dimensional system with practically no loss of information.

In our case, we use PCA only to tell us how many neurons we need to preserve in each layer. The retraining process applied after pruning will adjust the weights so that the output of the pruned layer is equivalent to computing the original neurons and then applying the linear transformation dictated by the PCA. In other words, the original n-dimensional output is not needed, and the pruned network is expected to produce the same results as if the original n-dimensional output was computed and the PCA transformation was applied afterwards.

The steps to apply the PCA-based pruning are as follow. First, we generate a trace of the outputs of the nodes of each fully-connected layer (outputs are taken after the activation function). A subset of the training dataset can be used to generate the trace. In our experiments, we use around 1% of the training set of each DNN to generate these traces. Then, we apply the PCA to the trace, which gives us the variance coefficients in the transformed coordinate system. Next, we compute how many of the lower-variance components can be removed while
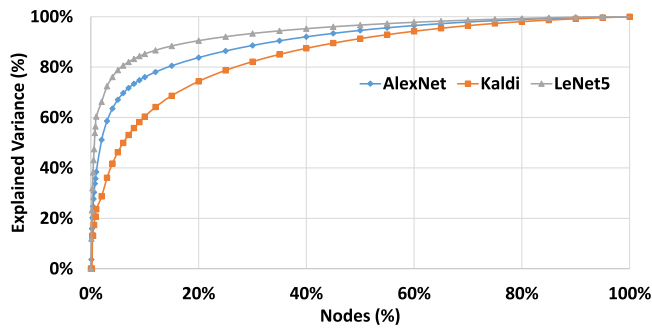
Fig. 6. Cumulative variance of a sample layer of AlexNet, Kaldi and LeNet5.

Fig. 7. Example of the weights distribution translation pruning problem.

still keeping 95% of the original variance. The number of remaining components is the number of neurons that we keep for this layer. Which nodes to keep and which are removed is irrelevant as demonstrated in the previous section, so they are randomly chosen. Once we determine the number of neurons in all layers, a single retraining of the pruned network is performed.

PCA cannot be directly applied to convolutional layers because a full feature map is considered as a single node which generates a volume of data. Since removing entire feature maps also have a high impact in accuracy we decided not to apply this step to convolutional layers. Note that previously proposed schemes for node pruning are also very inefficient for convolutional layers for the same reason.

Fig. 6 shows the cumulative variance of a sample fully-connected layer of AlexNet, Kaldi and LeNet5. For the sample layer of AlexNet, we can see that 50% of the nodes keep 95% of the original information, so 50% can be pruned. For the sample layer of Kaldi, we can see that 60% of the nodes keep 95% of the original information, so 40% can be pruned. Finally, for the sample layer of LeNet5, the benefits are much higher since we can remove 70% of the nodes while still keeping 95% of the information.

4.2. Pruning of connections through DIRIE

The second step of our method consists of pruning connections through the Distribution Invariant Relative Importance Estimation (DIRIE) metric after removing the redundant neurons. The rationale behind this approach is the following. The node pruning performed in the first step gives us a network with the minimal number of neurons in each fully-connected layer, while keeping the original accuracy. In the resulting topology there is no opportunity to remove further neurons; however, some of the connections may still have minor impact and can be removed to further reduce the size of the network. The obvious case are connections whose weight is zero. They clearly can be ignored without affecting the output. We could also remove all connections whose weight is close to zero, as the near-zero pruning [1] does. However, the importance of a connection is not necessarily related to how close to zero its weight is. For instance, a weight of 0.1 would be unimportant if the rest of the weights for the same neuron are in the order of 1 or greater, but will be important if the rest of the weights are similar or smaller. In a similar manner, a connection with a weight very different to zero, say for instance 10, will be unimportant if the rest of the connections of the same neuron are in the order of 100.

We propose to measure the importance of a connection as a function of the absolute value of its weight and the average absolute value of all the incoming weights of the same neuron. In other words, a connection is considered unimportant if the magnitude of its weight is small compared with the other weights of the same node. This step is applied to all layers including convolutionals where a full feature map is considered as a single node.

A first idea to measure the importance of a connection would be to compute the ratio of the absolute value of its weight to the average
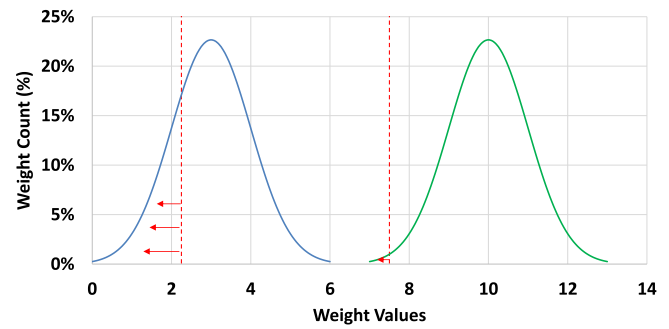
absolute value of all weights of the same neuron. This works relatively well if the distributions of the weights are centered around zero for all neurons. Besides, this metric is insensitive to scaling. That is, if all weights are multiplied by a given constant, the pruning scheme would still affect the very same weights. However, this metric is not insensitive to displacements (translations). That is, if we have another neuron whose weights are about the same but with an added offset (i.e, each weight of the new neuron results from adding a constant to a different weight of the other neuron), this metric would result in a different pruning, in spite of the fact that the weight distribution of the two neurons have exactly the same shape, one being displaced by a constant with respect to the other. Fig. 7 illustrates the translation problem with an example of two distributions of weights. Assuming a threshold of 75% of the mean, the weights on the left side of the red line would be pruned in each case. We can see that the number of weights pruned for the distribution centered on three would be much higher than for the distribution centered on ten, although the only difference between them is a translation of seven.

We want a metric to measure the importance of a connection that is insensitive to translation and scaling of the weights, hence, invariant to the weights distribution. To this end, we first take the absolute value of all weights. Then, for each neuron, we subtract the minimum absolute value of the weights of this node to the rest of the weights of that node. Finally, for each node we compute the mean of the resulting values and remove the connections whose value is smaller than 75% of the mean.

An observation to make is that after applying the pruning of connections the resulting network model will be sparse. Executing a sparse model is normally less efficient than a dense model so the pruning ratios achieved by this step must be significant to compensate for this penalty.

To summarize, the proposed scheme consists of two steps. First, we perform a node pruning based on a PCA analysis, to keep the minimum number of nodes that generate practically the same information as the original ones. Then, we prune the remaining connections based on a novel metric named DIRIE, removing those connections whose weight has an absolute value that is small compared with the rest of the connections of the same node. Retraining is applied only once after each of the two steps.

5. Evaluation methodology

Our goal is to prove that our pruning scheme, PCA+DIRIE, provides pruning ratios that are similar or better than some of the most popular schemes in spite of not being iterative, unlike most of the previous schemes, which makes them very costly or impractical. The schemes implemented and used for comparison are the **Baseline** where no pruning is applied, the **Near Zero Weights** using the connection pruning method described in Section 2.1, the **Input Weights Norm** which applies the node pruning method described in Section 2.2, the **Similarity** node pruning method described in Section 2.3, and the **Random Weights** and **Random Nodes** which are simplistic methods

that randomly chooses connections or nodes to prune given a target percentage. We do not include results for Scalpel (Section 2.4) and PCA Pruning (Section 2.5) since they perform worse than the above schemes.

To evaluate the pruning ratios achieved by PCA+DIRIE we apply it on five state-of-the-art DNNs for three different application domains: one for acoustic scoring in speech recognition, three for image classification and one for machine translation. We use the DNNs shown in Table 1 plus two more recent and larger DNN models that are not shown in the table due to their complexity in terms of number of layers. The DNN for acoustic scoring is the one included in the Kaldi [22] toolkit, a popular framework for speech recognition. We have ported it to Tensorflow [27] and trained it with the Librispeech [23] dataset that contains 100 h of speech. To evaluate LeNet5, we use images of digits from the MNIST [24] dataset. In addition, we employ AlexNet [14] in its second version [15] trained on ImageNet [17]. Finally, we evaluate the ResNet-50 [16] and the Transformer [28] models from the Tensorflow Model Garden [29] of Google, using the ImageNet dataset for image classification and the WMT16 dataset for translating from English to German (EN-DE), respectively. All the networks and pruned models have been implemented in Tensorflow. For all the DNNs, we employ the whole test or validation dataset to obtain the accuracy.

Regarding the training and retraining of the DNNs, all the baseline DNN models have been trained from scratch using the same hyperparameters provided in the papers where the models were originally presented. The retraining performed after each of the steps is conducted as follow. After the PCA node pruning, the models are trained from scratch using the same hyper-parameters of the baseline, but adding static masks at the outputs of each FC layer where the pruning is applied, which guarantees that the outputs of the pruned nodes are always zero even during the training. Then, after the pruning with DIRIE is applied, the DNNs are retrained starting from the previously trained model of the PCA step and, again, the same hyper-parameters are used, including another set of static masks that multiply the weights to ensure that the pruned weights stay at zero during all the retraining.

6. Results

This section evaluates the proposed PCA+DIRIE pruning scheme, described in Section 4, on different DNNs in terms of accuracy and amount of pruning. First, we compare our method with some of the most popular pruning strategies that have been previously proposed. Then, we discuss the scalability of PCA+DIRIE by showing experimental results on large and recent DNN models, including a brief quantitative comparison against some of the latest pruning methods. Finally, we present a sensitivity analysis of our scheme.

The main benefit of PCA+DIRIE is the time required to complete the pruning. For instance, AlexNet takes around 3 days to finish a retraining step on a GTX 1080 GPU. PCA+DIRIE requires two retraining steps so it will take 2x the retraining time, i.e. less than a week for AlexNet. On the other hand, for the methods that require exploring multiple configurations per layer such as the Near-Zero pruning, finding the appropriate pruning percentages for all layers requires n^l retraining steps, being n the number of configurations analyzed per layer and l the number of layers. Since AlexNet has eight layers, even for a very low value of n such as 3, the pruning process would take around 54 years in a high-end GPU, or half a year in a farm of 100 GPUs. Considering that current DNNs, such as Resnet152 or Densenet201, include hundreds of layers the applicability of previous methods may not be feasible. Note that future pruning methods, or even some of the latest methods proposed, may have a smaller search space and, hence, require less training time.

Table 2 shows the pruning effectiveness for the Kaldi DNN. For each pruning scheme we report the maximum pruning we could achieve with negligible accuracy loss (less than 0.25% in all cases). We can see that our method achieves the highest degree of pruning, resulting

Table 2

Accuracy (WER) and percentage of weights and computations removed by different pruning schemes for the Kaldi DNN on Librispeech.

Pruning method	WER(%)	Weights Pruned(%)	FLOPS Removed(%)
Baseline	10.04	0	0
Near Zero Weights	10.18	60	60
Random Weights	10.27	40	40
Input Weights Norm	10.24	40	40
Similarity	10.15	40	40
Random Nodes	10.15	40	40
PCA+DIRIE	10.28	70	70

Table 3

Accuracy (Top-1) and percentage of weights and computations removed by different pruning schemes for LeNet5 on MNIST.

Pruning method	Top-1(%)	Weights Pruned(%)	FLOPS Removed(%)
Baseline	99.34	0	0
Near Zero Weights	99.33	80	80
Random Weights	99.31	60	60
Input Weights Norm	99.37	87	20
Similarity	99.4	87	20
Random Nodes	99.29	87	20
PCA+DIRIE	99.41	79	52

Table 4

Accuracy (Top-1) and percentage of weights and computations removed by different pruning schemes for AlexNet on ImageNet.

Pruning method	Top-1 (%)	Weights Pruned(%)	FLOPS Removed(%)
Baseline	57.48	0	0
Near Zero Weights	56.55	80	87
Random Weights	56.46	50	63
PCA+DIRIE	59.7	67	51

in a 70% reduction of the weights and 70% reduction of the number of computations. The next best scheme is the near-zero pruning, which achieves a 60% reduction in both weights and computations. Since in Kaldi all the layers are fully-connected, the reduction in weights and the reduction in computations is the same. Note that our PCA+DIRIE scheme only requires to retrain the DNN twice whereas for the other methods we have to carry out an iterative search to find the maximum percentage of pruning with negligible accuracy loss, and the DNN has to be retrained for each pruning percentage.

Table 3 shows the results for the LeNet5 DNN. In this case, accuracy is measured as the top-1 so higher is better. Unlike Kaldi, in LeNet5 there are both convolutional and fully-connected layers. Most computations come from the convolutional layers while most of the weights are due to the fully-connected layers. Since some node pruning methods such as the similarity pruning can only be applied to fully-connected layers, they achieve a significant reduction in weights, but quite moderate in computations. We can observe that PCA+DIRIE prunes 79% of the weights and 52% of the computations, which is the second best in terms of weight and computation reduction, only slightly below the near-zero.

Table 4 shows the results for AlexNet. We can observe that PCA+DIRIE prunes 67% of the weights and 51% of the computations. It does not prune the convolutionals as much as the previous heuristics so the reduction in computations is lower, but it achieves higher accuracy.

We have also evaluated our Distribution Invariant Relative Importance Estimation (DIRIE) connection pruning metric alone to demonstrate that it is more effective than Near-Zero pruning. Table 5 shows the results for 70%–90% of pruning of the Kaldi DNN. We can see that DIRIE achieves better accuracy with the same amount of pruning and, hence, taking into account the importance of the connections relative to each node is more effective than just considering the magnitude of all the weights on each layer.

Table 5

Link Pruning Comparison for the Kaldi DNN on Librispeech. (Baseline WER = 10.04%).

Weights Pruned (%)	Near-Zero WER (%)	DIRIE WER (%)
70	10.67	10.53
80	11.39	11.10
90	14.62	13.56

Table 6

Accuracy (Top-1) and percentage of weights and computations removed by different pruning schemes for ResNet-50 on ImageNet. M75% and M100% refer to the threshold of the mean employed in the DIRIE step.

Pruning method	Baseline Top-1(%)	Pruned Top-1(%)	Accuracy Drop(%)	Weights Pruned(%)	FLOPS Removed(%)
ThiNet [30]	72.88	72.04	0.84	–	36.7
NISP [31]	–	–	0.89	–	44.0
CP [32]	–	–	1.40	–	50.0
SFP [33]	76.15	62.14	14.01	–	41.8
FPGM [34]	76.15	75.59	0.56	–	42.2
ASFP [35]	76.15	75.53	0.62	–	41.8
LW-DNA [13]	76.72	**77.00**	**0.00**	7.1	9.4
MetaPruning [36]	76.72	76.20	0.52	–	27.2
AutoSlim [37]	76.72	76.00	0.72	9.6	27.2
DIRIE (M75%)	**76.80**	76.80	**0.00**	48.0	49.0
DIRIE (M100%)	**76.80**	76.60	0.20	**60.0**	**61.0**

6.1. Scalability analysis

In order to prove that PCA+DIRIE can be effectively applied to any type of DNN, and scale properly to large and recent models, in this subsection we show the evaluation of our scheme for ResNet-50 [16] on ImageNet and the Transformer [28] model on the WMT 2014 English-to-German translation task. Both networks are state-of-the-art models for image classification and machine translation, respectively. ResNet is composed of over 50 CONV layers, while the Transformer is mainly composed of attention layers and feed-forwards which include over 90 FC layers in total. In the case of ResNet-50, we only apply the DIRIE step of our pruning scheme since the PCA cannot be used for CONV layers. In addition, we compare our results against recent works pruning ResNet-50 to show the effectiveness of our method. In contrast to the previous section, note that we have not implemented and evaluated these pruning methods but the results have been directly taken from the corresponding papers for comparison purposes. On the other hand, we apply both pruning steps in the Transformer network to demonstrate the full applicability and scalability of our method on one of the latest and largest DNN model architectures.

Table 6 shows the pruning effectiveness of DIRIE for the ResNet-50 model. For each pruning scheme, we report the accuracy (Top-1) of the baseline model, the accuracy of the pruned model, the accuracy drop in absolute terms, the percentage of weights pruned, and the percentage of computations removed. We have performed experiments with DIRIE corresponding to different thresholds of the mean of the weights of each node (i.e. M75% and M100%). We can see that our method achieves the highest degree of pruning, resulting in a 60% reduction of the weights and 61% reduction of the number of computations with negligible accuracy loss. The next best scheme in terms of both pruning and accuracy drop is the FPGM pruning, which achieves a 42.2% reduction in computations. Note that our DIRIE scheme only requires to retrain the DNN once whereas for most of the other methods there may be multiple retrainings.

Table 7 shows the pruning effectiveness of PCA+DIRIE for the Transformer model. We show the results obtained for a couple of experiments corresponding to the PCA step with a threshold of 95% for the coefficient of variance (CV) to remove nodes, and the DIRIE step on top of the PCA with a threshold of 75% of the mean of the weights of each node to remove connections. For each experiment we report the accuracy (BLEU Score Cased), the accuracy drop in absolute terms, and the percentage of weights pruned. We can see that our method results in a 50.8% reduction of the weights with negligible impact in accuracy.

Table 7

Accuracy (BLEU Score Cased) and percentage of weights removed by different pruning schemes for the Transformer model on the WMT 2014 English-to-German translation task.

Pruning method	BLEU Score Cased	Accuracy drop	Weights pruned(%)
Baseline	27.94	0.00	0.0
PCA (CV95%)	27.66	0.28	15.0
PCA+DIRIE (M75%)	27.88	0.06	50.8

Table 8

Kaldi on Librispeech results after the PCA step (first step) of the proposed pruning scheme using different coefficients of variance. (Baseline WER = 10.04%).

CV (%)	WER (%)	Weights Pruned(%)	FLOPS Removed(%)
99	10.00	15	15
95	10.24	45	45
90	10.62	64	64

6.2. Sensitivity analysis

The proposed pruning scheme uses two thresholds to control the amount of pruning and the loss of accuracy. Our key goal is to find values for these parameters that achieve high efficiency, i.e. large amount of pruning with negligible accuracy loss, for a wide range of DNNs. Therefore, the user does not have to manually tune these thresholds for each specific DNN, as it happens with other pruning schemes.

In our scheme, the first step (node pruning) is applied by keeping only as many neurons as PCA components are needed to represent 95% of the original information (coefficient of variance). In the second step, connections are pruned based on its relative importance compared to the other connections of the same neuron.

We performed a sensitivity analysis using the Kaldi DNN since the accuracy of Kaldi is highly sensitive to model changes and the model size is reasonable. The parameters obtained from this analysis are used for all the networks, achieving the results shown in Section 6, which confirms that these parameters work well for different networks.

Table 8 shows the results for the Kaldi DNN after applying the first step of our pruning method using different thresholds for the coefficient of variance (CV). The accuracy of Kaldi is measured as Word Error Rate (WER), so lower is better. As it can be seen, with 99% of variance we can maintain the same accuracy or even slightly better, but the percentage of pruning is dramatically reduced to only 15%. In contrast, if we use 90% of variance, the pruning is quite high (64%) but the accuracy is slightly affected (0.58% loss). Finally, if we use 95% of variance the accuracy is almost the same (only 0.2% of loss) and the percentage of pruning is 45%. Therefore, we use 95% of coefficient of variance as the by default value for our scheme. In case that a small loss of accuracy could be assumed, 90% of CV could be a good choice in order to achieve a higher degree of pruning.

Table 9 shows the results after the second step of our pruning method using different thresholds for the percentage of the mean (computed as described in Section 4.2), which determines when a connection is pruned for the Kaldi DNN. We can conclude that 75% is the most adequate threshold. A higher threshold can prune more links but looses some accuracy, whereas a lower threshold has about the same accuracy but is less effective at pruning.

In summary, we set the by default pruning configuration to work with 95% for the coefficient of variance to remove nodes and 75% of the mean of the weights of each node to remove connections. Using this configuration PCA+DIRIE achieves pruning percentages of 70%(Kaldi), 79%(LeNet5), 67%(AlexNet), 48%(ResNet-50) and 51%(Transformer) with negligible accuracy loss.

Table 9
Kaldi on Librispeech results after the unimportant connections pruning step (second step) of the proposed pruning scheme for different thresholds of the mean. (Baseline WER = 10.04%).

CV(%)	Mean(%)	WER(%)	Weights Pruned(%)	FLOPS Removed(%)
95	100	10.62	77	77
95	75	10.28	70	70
95	50	10.26	63	63

7. Related work

DNNs have become very popular in a wide range of environments and devices, from large data centers and high performance computers [38,39] to mobile devices [40,41]. DNNs are computationally and memory intensive, and consume a significant amount of energy. Therefore, custom architectures with optimizations such as pruning can provide important benefits. Some of the most popular applications of MLPs are speech recognition and machine translation [42,43], whereas CNNs are commonly used for image classification [44].

DNN Optimizations. Proposals for reducing the memory footprint and/or computations of DNNs include clustering [45], linear quantization [46] and pruning [47–51]. Clustering uses methods such as K-means to reduce the number of different weights to K centroids. Each weight is then substituted by an index that corresponds to the closest centroid. Since the weights tend to be very similar, the number of centroids per layer can be kept relatively low (in the order of 16-256), which significantly reduces the storage requirements and memory bandwidth for the weights. However, computations still have to be performed in floating point by using the centroids and the total amount of computations is not reduced at all. Linear quantization maps each value, either weights or inputs, to a discrete set of values distributed over the whole range of possible values. Values are replaced by indexes, which identify the discrete set of values, and reduces their storage requirements. Unlike clustering, since the quantization is linear, most computations can be done by using directly the integer indexes rather than the corresponding floating point values. The amount of computations is not reduced, but most computations are simpler since they operate on integer numbers rather than floating point. Therefore, quantization is particularly efficient in DNN accelerators to not only reduce the model size but also the energy consumption of most computations.

Iterative Pruning. As described in Section 2 there is a large number of iterative methods to prune DNNs. Some methods remove connections depending on the weights' values [1], others remove nodes taking into account the weights of each node [9,10]. PCA-based methods for fast pruning have been previously proposed [7,8,52–56]. However, they just project the weights and inputs using the principal components and do not apply retraining. The amount of pruning achieved with these PCA-based methods is extremely limited for modern DNNs. Most of these pruning methods are applied after training a baseline non-pruned network. On the other hand, some methods perform pruning during the training phase by learning the connections or nodes that are redundant [3,57,58]. We show in Section 3 that most of these previous methods are impractical due to the large number of parameters that have to be tuned for each DNN. Our proposal is inspired in previously proposed PCA-based techniques and weight pruning schemes and we show that it is highly effective and, more importantly, practical for a wide range of DNNs.

Non-Iterative Pruning. Pruning large and complex neural networks while maintaining their performance is often desirable due to the reduced space and time complexity. As described above, the majority of existing methods perform pruning within an iterative optimization procedure, using heuristically designed pruning schedules that require additional hyperparameters and many expensive prune-retrain cycles, undermining their utility and making them non-trivial to extend to new

architectures and tasks. Similar to this work, in recent years there has been some studies proposing non-iterative pruning methods to reduce the number of training iterations to just a single experiment (i.e. one-shot) with no additional parameters, or very few, simplifying the pruning procedure, and increasing the versatility to apply these methods to different neural network architectures. AutoSlim [37] performs channel pruning of CNNs in a single pass. They train a single slimmable network to approximate the network accuracy of different channel configurations, and greedily slim the layer with minimal accuracy drop. OPQ [59] performs pruning and quantization at the same time by analytically solving the compression allocation problem with pre-trained weight parameters only. OTO [60] is a framework for training and pruning DNNs at once. They partition the parameters of DNNs into zero-invariant groups, and promote zero-groups by formulating a structured-sparsity optimization problem along a new optimization algorithm to solve it. The work in [61] proposes SNIP, a method that prunes a given DNN once at initialization prior to training. SNIP employs a saliency criterion based on connection sensitivity to identify important connections in a data-dependent way before training. Specifically, SNIP measures the importance of the connections based on their influence on the loss function at a variance scaling initialization. Given the desired sparsity level, redundant connections are pruned once prior to training, and then the sparse pruned network is trained in the standard way. Compared to PCA+DIRIE, SNIP is heavily reliant on the network initialization of the weights, as well as the batch of data employed to measure the saliency score. Moreover, SNIP still requires the user to manually specify the overall target pruning percentage, which may end up requiring more than a single experiment to find a reasonable value, and has not demonstrated its scalability on bigger and more complex DNNs.

8. Conclusions

In this paper, we show that many of the popular DNN pruning schemes require that multiple parameters be configured by a trial and error process, often resulting in an exponential number of configurations to be evaluated, which may be impractical since each experiment requires a retraining of the network, which is an extremely costly operation. We propose the PCA+DIRIE pruning scheme that overcomes this weakness. PCA+DIRIE consists of two steps, one to remove redundancy at the neuron level, and another to remove redundancy at the connection level. The first step is based on a Principal Component Analysis (PCA) to remove nodes while the second step removes unimportant connections based on a novel metric that measures the relative importance of each connection (DIRIE). The proposed scheme requires only two retraining operations and achieves results similar or even better than state-of-the-art iterative pruning methods.

Declaration of competing interest

All (UPC), Alessandro Vallero (Politecnico di Torino), Alessandro Savino (Politecnio di Torino), Stefano Di Carlo (Politecnico di Torino), Riccardo Mariani (Intel), Athanasios Chatzidimitriou (University of Athens), Manolis Kaliorakis (University of Athens), Maha Kooli (CEA Leti), Alberto Bosio (Ecole Centrale de Lyon), Dimitris Gizopoulos (University of Athens), Giorgio Di Natale (LIRMM), Juan Luis Aragón (University of Murcia), Lieven Eeckhout (Ghent University), Wim Heirman (Ghent University), David Kaeli (Northeastern University), Rakesh Kumar (NTNU), Jose Cano (University of Glasgow), Sudhanshu Jha (ARM), Gaurang Upasani (Intel), Aleksandar Brankovic (Intel), Shrikanth Ganapathy, Pedro Lopez (Microsoft), Stefan Bieschewski, Govind Shenoy, Carlos Madriles (ARM), Abhishek Deb (Nvidia), Indu Bhagat, Marc Lupon, Javier Lira, Rakesh Ranjan (Nvidia), Alex Aleta, Fernando Latorre (ARM), Eduardo Quiñones (Barcelona Supercomputing Center), Josep Maria Codina (Amazon), Pedro Chaparro, Carlos Molina (Universitat Rovira i Virgili), Enric Gibert,

Jaume Abella, (Barcelona Supercomputing Center), Pedro Marcuello, Jesus Sanchez (Microsoft), Jose Gonzalez (ARM), Luis Diaz de Cerio, Reza Yazdani (Microsoft), Hamid Tabani (Barcelona Supercomputing Center), Gem Dot (ARM), Martí Torrents (Huawei), Martí Anglada (Huawei), Albert Segura (Apple)

Acknowledgments

References

[1] S. Han, J. Pool, J. Tran, W.J. Dally, Learning both weights and connections for efficient neural networks, in: Advances in Neural Information Processing Systems (NIPS), 2015, pp. 1135–1143.

[2] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M.A. Horowitz, W.J. Dally, EIE: Efficient inference engine on compressed deep neural network, in: ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), 2016, pp. 243–254.

[3] J. Yu, A. Lukefahr, D. Palframan, G. Dasika, R. Das, S. Mahlke, Scalpel: Customizing DNN pruning to the underlying hardware parallelism, in: ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA), 2017, pp. 548–560.

[4] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S.W. Keckler, W.J. Dally, SCNN: An accelerator for compressed-sparse convolutional neural networks, in: ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA), 2017, pp. 27–40.

[5] Y. Li, S. Gu, L.V. Gool, R. Timofte, Learning filter basis for convolutional neural network compression, in: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), 2019, pp. 5623–5632.

[6] X. Zhang, J. Zou, K. He, J. Sun, Accelerating very deep convolutional networks for classification and detection, IEEE Trans. Pattern Anal. Mach. Intell. (TPAMI) 38 (10) (2016) 1943–1955.

[7] A.U. Levin, T.K. Leen, J.E. Moody, Fast pruning using principal components, in: Advances in Neural Information Processing Systems (NIPS), 1994, pp. 35–42.

[8] D. Opitz, Analyzing the structure of a neural network using principal component analysis, in: Proceedings of International Conference on Neural Networks (ICNN), 1997, pp. 254–259.

[9] T. He, Y. Fan, Y. Qian, T. Tan, K. Yu, Reshaping deep neural network for fast decoding by node-pruning, in: IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2014, pp. 245–249.

[10] S. Srinivas, R.V. Babu, Data-free parameter pruning for deep neural networks, ArXiv, 2015.

[11] Z. Liu, M. Sun, T. Zhou, G. Huang, T. Darrell, Rethinking the value of network pruning, ArXiv, 2018.

[12] Y. Li, S. Gu, K. Zhang, L. Van Gool, R. Timofte, Dhp: Differentiable meta pruning via hypernetworks, in: European Conference on Computer Vision (ECCV), 2020, pp. 608–624.

[13] Y. Li, W. Li, M. Danelljan, K. Zhang, S. Gu, L. Van Gool, R. Timofte, The heterogeneity hypothesis: Finding layer-wise differentiated network architectures, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2021, pp. 2144–2153.

[14] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, Advances in Neural Information Processing Systems (NIPS) 25 (2012) 1097–1105.

[15] A. Krizhevsky, One weird trick for parallelizing convolutional neural networks, ArXiv, 2014.

[16] K. He, X. Zhang, S. Ren, J. Sun, Deep Residual Learning for Image Recognition, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770–778.

[17] ImageNet, Imagenet dataset, 2009, Online; https://image-net.org/. (Accessed 02 July 2021).

[18] G. Huang, Z. Liu, L. Van Der Maaten, K.Q. Weinberger, Densely connected convolutional networks, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 4700–4708.

[19] J. Hu, L. Shen, G. Sun, Squeeze-and-excitation networks, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018, pp. 7132–7141.

[20] R. Rojas, Neural Networks: A Systematic Introduction, Springer Science & Business Media, 2013.

[21] X. Zhang, J. Trmal, D. Povey, S. Khudanpur, Improving deep neural network acoustic models using generalized maxout networks, in: IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2014, pp. 215–219.

[22] D. Povey, Kaldi software, 2013, Online; http://kaldi-asr.org/. (Accessed 02 July 2021).

[23] V. Panayotov, G. Chen, D. Povey, S. Khudanpur, Librispeech: An ASR corpus based on public domain audio books, in: IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2015, pp. 5206–5210.

[24] Y. LeCun, MNIST Dataset, 1998, Online; http://yann.lecun.com/exdb/mnist/. (Accessed 02 July 2021).

[25] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, ArXiv, 2014.

[26] M.E. Tipping, C.M. Bishop, Probabilistic principal component analysis, J. R. Stat. Soc. Ser. B Stat. Methodol. 61 (3) (1999) 611–622.

[27] Google, Tensorflow framework, 2015, Online; https://www.tensorflow.org/. (Accessed 02 July 2021).

[28] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, in: Advances in Neural Information Processing Systems (NIPS), 2017, pp. 5998–6008.

[29] H. Yu, C. Chen, X. Du, Y. Li, A. Rashwan, L. Hou, P. Jin, F. Yang, F. Liu, J. Kim, J. Li, Tensorflow model garden, 2020, Online; https://github.com/tensorflow/models. (Accessed 02 July 2021).

[30] J.-H. Luo, J. Wu, W. Lin, Thinet: A filter level pruning method for deep neural network compression, in: IEEE International Conference on Computer Vision (ICCV), 2017, pp. 5058–5066.

[31] R. Yu, A. Li, C.-F. Chen, J.-H. Lai, V.I. Morariu, X. Han, M. Gao, C.-Y. Lin, L.S. Davis, Nisp: Pruning networks using neuron importance score propagation, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018, pp. 9194–9203.

[32] Y. He, X. Zhang, J. Sun, Channel pruning for accelerating very deep neural networks, in: IEEE International Conference on Computer Vision (ICCV), 2017, pp. 1389–1397.

[33] Y. He, G. Kang, X. Dong, Y. Fu, Y. Yang, Soft filter pruning for accelerating deep convolutional neural networks, in: International Joint Conference on Artificial Intelligence (IJCAI), 2018, pp. 2234–2240.

[34] Y. He, P. Liu, Z. Wang, Z. Hu, Y. Yang, Filter pruning via geometric median for deep convolutional neural networks acceleration, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2019, pp. 4340–4349.

[35] Y. He, X. Dong, G. Kang, Y. Fu, C. Yan, Y. Yang, Asymptotic soft filter pruning for deep convolutional neural networks, IEEE Trans. Cybern. 50 (8) (2020) 3594–3604.

[36] Z. Liu, H. Mu, X. Zhang, Z. Guo, X. Yang, K.-T. Cheng, J. Sun, Metapruning: Meta learning for automatic neural network channel pruning, in: IEEE International Conference on Computer Vision (ICCV), 2019, pp. 3296–3305.

[37] J. Yu, T. Huang, Autoslim: Towards one-shot architecture search for channel numbers, ArXiv, 2019.

[38] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, O. Temam, DaDianNao: A machine-learning supercomputer, in: 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2014, pp. 609–622.

[39] W. Xiong, J. Droppo, X. Huang, F. Seide, M. Seltzer, A. Stolcke, D. Yu, G. Zweig, Achieving human parity in conversational speech recognition, ArXiv, 2016.

[40] J. Jin, V. Gokhale, A. Dundar, B. Krishnamurthy, B. Martini, E. Culurciello, An efficient implementation of deep convolutional neural networks on a mobile coprocessor, in: IEEE 57th International Midwest Symposium on Circuits and Systems (MWSCAS), 2014, pp. 133–136.

[41] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, O. Temam, ShiDianNao: Shifting vision processing closer to the sensor, in: ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA), 2015, pp. 92–104.

[42] G.E. Dahl, T.N. Sainath, G.E. Hinton, Improving deep neural networks for LVCSR using rectified linear units and dropout, in: IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2013, pp. 8609–8613.

[43] J. Zhang, C. Zong, Deep neural networks in machine translation: An overview, IEEE Intell. Syst. 30 (5) (2015) 16–25.

[44] M. Alwani, H. Chen, M. Ferdman, P. Milder, Fused-layer CNN accelerators, in: 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2016, pp. 1–12.

[45] Y. Gong, L. Liu, M. Yang, L. Bourdev, Compressing deep convolutional networks using vector quantization, ArXiv, 2014.

[46] N.P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, et al., In-datacenter performance analysis of a tensor processing unit, in: ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA), 2017, pp. 1–12.

[47] T.-Y. Hsiao, Y.-C. Chang, H.-H. Chou, C.-T. Chiu, Filter-based deep-compression with global average pooling for convolutional networks, J. Syst. Archit. (JSA) 95 (2019) 9–18.

[48] D. Moolchandani, A. Kumar, S.R. Sarangi, Accelerating CNN inference on ASICs: A survey, J. Syst. Archit. (JSA) 113 (2021) 101887.

[49] S. Mittal, Vibhu, A survey of accelerator architectures for 3D convolution neural networks, J. Syst. Archit. (JSA) 115 (2021) 102041.

[50] S. Pattanayak, S. Nag, S. Mittal, CURATING: A Multi-objective based pruning technique for CNNs, J. Syst. Archit. (JSA) 116 (2021) 102031.

[51] M. Xia, Z. Huang, L. Tian, H. Wang, V. Chang, Y. Zhu, S. Feng, SparkNoC: AN energy-efficiency FPGA-based accelerator using optimized lightweight CNN for edge computing, J. Syst. Archit. (JSA) 115 (2021) 101991.

[52] W. Wen, C. Wu, Y. Wang, Y. Chen, H. Li, Learning structured sparsity in deep neural networks, in: Advances in Neural Information Processing Systems (NIPS), vol. 29, 2016, pp. 2074–2082.

[53] L. Mauch, B. Yang, Selecting optimal layer reduction factors for model reduction of deep neural networks, in: IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2017, pp. 2212–2216.

[54] P. Thomas, M.-C. Suhner, A new multilayer perceptron pruning algorithm for classification and regression applications, Neural Process. Lett. 42 (2) (2015) 437–458.

[55] A. Engelbrecht, A new pruning heuristic based on variance analysis of sensitivity information, IEEE Trans. Neural Netw. (TNNLS) 12 (6) (2001) 1386–1399.

[56] R.A. Koene, Y. Takane, Discriminant component pruning: Regularization and interpretation of multilayered backpropagation networks, Neural Comput. 11 (3) (1999) 783–802.

[57] Y. Guo, A. Yao, Y. Chen, Dynamic network surgery for efficient DNNs, in: Advances in Neural Information Processing Systems (NIPS), 2016, pp. 1387–1395.

[58] T.-J. Yang, Y.-H. Chen, V. Sze, Designing energy-efficient convolutional neural networks using energy-aware pruning, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 5687–5695.

[59] P. Hu, X. Peng, H. Zhu, M.M.S. Aly, J. Lin, OPQ: Compressing deep neural networks with one-shot pruning-quantization, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, 2021, pp. 7780–7788.

[60] T. Chen, B. Ji, T. Ding, B. Fang, G. Wang, Z. Zhu, L. Liang, Y. Shi, S. Yi, X. Tu, Only train once: A one-shot neural network training and pruning framework, ArXiv, 2021.

[61] N. Lee, T. Ajanthan, P.H. Torr, Snip: Single-shot network pruning based on connection sensitivity, ArXiv, 2018.

Marc Riera received his B.S degree in Computer Engineering in 2013, his MS degree in MIRI: High Performance Computing in 2015, and his Ph.D. in Computer Architecture in 2020, all from Universitat Politècnica de Catalunya (UPC - BarcelonaTech). He joined the ARCO research group in 2014, and is currently a postdoctoral researcher at ARCO. His research interests focus on the area of Accelerator Architectures, Machine Learning and Near-Data Processing (NDP).

Jose-Maria Arnau received Ph.D. on Computer Architecture from the Universitat Politecnica de Catalunya (UPC) in 2015. He is a postdoctoral researcher at UPC BarcelonaTech and a member of the ARCO (ARchitecture and COmpilers) research group at UPC. His research interests include low-power architectures for cognitive computing, especially in the area of automatic speech recognition and object recognition.

Antonio González (Ph.D. 1989) is a Full Professor at the Computer Architecture Department of the Universitat Politècnica de Catalunya, Barcelona (Spain), and the director of the Architecture and Compilers research group. He was the founding director of the Intel Barcelona Research Center from 2002 to 2014. His research has focused on computer architecture and compilers, with a special emphasis on cognitive computing systems and graphics processors in recent years. He has published over 380 papers, and has served as associate editor of five IEEE and ACM journals, program chair for ISCA, MICRO, HPCA, ICS and ISPASS, and general chair for MICRO and HPCA. He is a Fellow of IEEE and ACM.