Investigating the Impact of Pruning on Energy-Accuracy of Neural Networks

Thomas Fontanari*, Leandro M. G. Rocha*, Gustavo M. Santana*, Guilherme Paim*, Eduardo A. C. da Costa†, Sergio Bampi*
*Graduate Program on Microelectronics (PGMicro) - Universidade Federal do Rio Grande do Sul (UFRGS) Porto Alegre - Brazil
†Graduate Program on Electronic Engineering and Computing - Universidade Católica de Pelotas (UCPel) Pelotas - Brazil

Abstract—**Neural networks have been achieving state-of-the-art performance in various problems, such as image classification and natural language processing. They are, however, computationally expensive. With billions of parameters and requiring an extensive amount of arithmetic operations, energy consumption is, therefore, a hard requirement. One of the ways to approach the energy problem is through pruning, a commonly used technique in approximate computing that consists of zeroing parameters given a specific rules. In this paper, we aim to analyze the effects of pruning on energy consumption and network accuracy. We apply an automated gradual pruning (AGP) method to five different networks and test them using the MNIST and CIFAR10 datasets. Our results show that energy consumption can be substantially reduced while maintaining and even improving accuracy levels; outcomes that cannot be obtained through smaller analogous networks.**

Keywords—Neural networks; Pruning; Energy consumption.

I. Introduction

Neural networks (NNs) always required a considerable computational power to execute their algorithms quickly and efficiently [1]. Since the inception of this field, scientists sought techniques to both optimize the algorithms and improve the underlying hardware to overcome the inherent complexity of these algorithms. Deep neural networks (DNN) have recently achieved state-of-the-art performance in various tasks such as image classification [2] and natural language processing [3]. These results however come at the cost of increasingly higher computational power requirements and memory intensiveness. AlexNet [2] for instance performs over 6 billions multiply-and-accumulate (MAC) operations just to classify a single image. The end result is high energy consumption derived mainly from memory accesses, which hinders its use in embedded devices. In order to overcome this problem, different approaches attempt to reduce the size of the network model, e.g., by pruning or using some form of quantization [4].

State-of-the-art neural networks require millions of parameters to describe an accurate model for classification tasks. The majority of these parameters reside on the classifier, usually composed of fully-connected layers. However, some connections in a NN have minimal impact on the model accuracy. For instance, the AlexNet model requires more than 200 MB of storage whereas the VGG-16 network requires over 500 MB to store the network parameters [5].

Pruning is a technique that reduces the number of memory accesses and computations in a NN by increasing sparsity, i.e., the proportion of parameters in the network that are equal to zero. MAC operations involving zeroed parameters do not need to be computed, nor the parameters used need to be accessed. Therefore, this results in a lower energy consumption per classification metric. Moreover, it is also reasonable to expect effects in the accuracy of the networks. The first approaches to pruning suggested using second derivatives as a way of estimating the importance of any given weight [6]. Recently, however, magnitude-based pruning has been used since it is much easier to perform in larger networks [7], and achieves comparable results.

In this paper we explore the effects of magnitude-based pruning on energy consumption and network accuracy rate. We train a group of LeNet [8] networks - LeNet5, LeNet-300-100-10, LeNet-300-10, a variation of LeNet5 and a linear classifier using the MNIST and CIFAR10 datasets. By applying the automated gradual pruning method introduced in [9] to these pre-trained networks we achieve very high sparsity levels while maintaining the accuracy rate. These sparsity levels are used together with a model of an architecture to run the networks in order to estimate the reduction in energy consumption brought by pruning. Moreover, our results show that similar reductions cannot be achieved by analogous smaller but denser network models – that is, analogous networks possessing the same number of non-zero parameters as the sparse original versions –, since the layer structure resulting from pruning cannot be represented in a smaller fully-connected or convolutional-based NN.

The main contribution of this paper is an investigation about the pruning techniques in NNs and its impact on the energy-accuracy trade-off between five existing architectures, using two existing data-sets.

This work is organized as follows: Section II presents a background about NNs detailing the pruning techniques. Section III presents our experiments for energy-accuracy trade-off consumption considering relevant pruning effects. Finally, Section IV concludes the paper.

II. Background

This section begins by reviewing neural networks. We then review the concept of pruning focusing on the automated gradual pruning (AGP) algorithm, which was used in this work. Further information on neural networks can be found in [4].

A. Neural Networks

A neural network (NN) is a composition of a set of smaller units named neurons. A neuron produces an output y by applying a non-linear function $f(\vec{x})$ to a weighted sum of its inputs $\vec{x}$. That is, $y = f(\vec{x}^T \cdot \vec{w})$, where $\vec{w}$ is the weight vector. This is illustrated in Figure 1. The computed output, termed the activation of a neuron, is used as input to other neurons in the neural network.

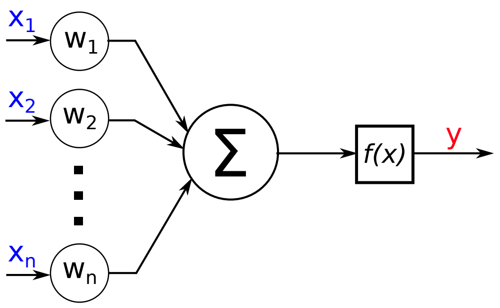

Fig. 1: Neuron used in neural networks.

Furthermore, a neural network is divided in layers, where each layer is composed of N neurons. Fully-connected layers are layers where each neuron has one connection to each neuron in the previous layer. This is illustrated in Figure 2, where a layer with 2 input neurons and 3 output neurons is shown.

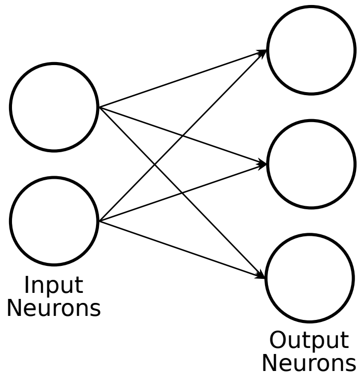

Fig. 2: Fully-connected layer. The arrows represent the weighted connections between the input and output neurons.

The first layer in a network is called the input layer, where the outputs are simply the inputs to the network. The last layer is named output layer and the outputs from its neurons are the neural network outputs. A DNN classifying gray scale images with 28x28 pixels, for instance, would have 784 neurons in its input layer and the output of each neuron would be the value of its corresponding pixel. Moreover, if there were 10 different classes of images, the network could have 10 neurons in its output layer, where the activation of each neuron could represent the probability of the input image belonging to one of the 10 classes. In a deep neural network there are also many layers between the input and output layers and they are thus called *hidden layers*.

The way in which the weights are arranged from the input neurons to the output neurons in a layer defines its type. Two commonly used types of layers are the fully connected (FC) and the convolutional layers, though many others exist. In a FC layer, there is a different connection between every input neuron and every output neuron, while a convolutional layer is characterized by an arrangement resembling the mathematical convolution operation.

A fully connected layer is shown in Figure 2. The value of its output neurons is given by

$$y_i = f(\sum_{j=1}^{M} w_{ij} x_j) \tag{1}$$

where y_i is the activation of the layer output neuron i, w_{ij} is the weight from neuron j to i, x_j is the activation of the input neuron j, and M is the number of neurons input neurons. Finally, f is a non-linear function, such as the Sigmoid or ReLU functions. In other words, we simply compute each neurons output given that their inputs are the activation of the input neurons.

B. Convolutional Layer

Designing a network using only fully-connected layers is not efficient as it does not scale well when the input size grows [10]. For instance, the first hidden layer of 3-layer MLP neural network using a 64×64 colored image as input would lead to 12288 parameters to describe the neural connection.

At a glance, restricting the number of connections among neurons arises as a promising alternative. The visual cortex in the human brain has several localized receptive fields whose neurons only produce spikes when there are stimuli with specific patterns or orientations [10]. Further, images are considered to be stationary, meaning that the statistical characteristics of a patch of that image will be the same as any other patch within the same image. Therefore, it is intuitive to think that a given local receptive field that identifies a feature – like a diagonal line – can be applied to several parts of an image.

A convolutional layer, as mentioned before, applies a convolution to its inputs. Its weights are therefore the elements of the kernel applied. Moreover, convolutional neural networks (CNN) are, in general, used when inputs are 2D images with multiple channels (e.g., red, green and blue components). We therefore consider a convolution between a kernel with dimensions KxK and a HxW image I, both with C channels, to be given by

$$(w * I)[x][y] = \sum_{c=1}^{C} \sum_{i=1}^{K} \sum_{j=1}^{K} w[c][i][j] \times I[c][Ux+i][Uy+j] \quad (2)$$

where U is the stride, which in a typical convolution is equal to 1. From it we note that x must be between 0 and $(W-K+U)/U$ while y must be between 0 and $(H-K+U)/U$, so that our image is indexed correctly. A convolutional layer can be said to perform M different convolutions, thus producing as output an image with M components, and its computation is given by

$$O[u] = f(w[u] * I[u]) \quad (3)$$

where u goes from 1 to M. Finally, a non-linear function f is applied to each output.

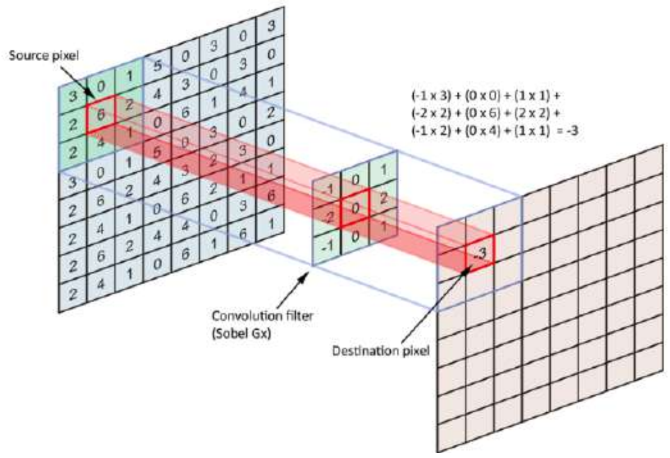
Source pixel
(-1 x 3) + (0 x 0) + (1 x 1) +
(-2 x 2) + (0 x 6) + (2 x 2) +
(-1 x 2) + (0 x 4) + (1 x 1) = -3
Convolution filter
(Sobel Gx)
Destination pixel

Fig. 3: Example a 3 × 3 2D convolution [11]

C. Pruning Techniques

Pruning a neural network results in setting some of its weights to zero, according to a given rule. In general, it is possible to greatly reduce the number of weights since the networks are usually over parameterized, thus containing a lot of redundancy [4]. Pruning techniques were first based on estimating the sensitivity of the error function to the removal of the weights [12]. The optimal Brain Damage [6] underlying idea was to use second derivatives of the loss function to estimate which weights could be removed. Overall, such techniques present better results than magnitude-based pruning [13], i.e., deleting a weight based on its magnitude. More recently, however, magnitude-based techniques have been brought back since they are much simpler to perform in large neural networks. In [7] a method is proposed in which magnitude pruning is applied in between training steps, thus allowing the network to adapt itself and achieving great reduction in the number of weights while maintaining the same accuracy. Furthermore, pruning can also be combined with different network compression techniques [5], or applied in a more structured way, such as by removing entire filters in convolutional layers [14]. Moreover, an energy-aware technique for pruning is described in [15] and shown to obtain interesting results.

In [9] the authors present a method for performing automated gradual pruning (AGP) in which sparsity levels for each neural network layer is increased at each step from an initial value s_i to s_f according to the equation:

$$s_t = s_f + (s_i - s_f)\left(1 - \frac{t - t_0}{n\Delta t}\right)^3 \quad (4)$$

where t_0 is the first training step in which pruning occurs, with a target sparsity level of $s_t = s_i$. n is the number of pruning steps and the pruning frequency Δt is the number of training steps between each pruning. In order to achieve the desired layer sparsity, the weights with the smallest magnitudes are pruned.

The reasoning behind the equation, as explained by the authors, is that since the equation incurs in a rapidly increasing sparsity in the first steps and a slower increase at the end phase, the great number of redundant weights present in the initial phase will be promptly removed, while the important weights remaining in the final phases will be kept. Figure 4 illustrates this process, where the pruning occurs at every training step, during 10 steps, aiming at a final sparsity of 80%.

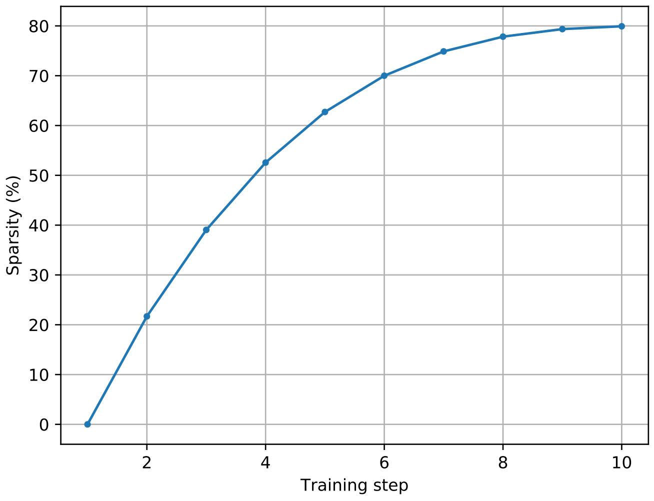

Fig. 4: Sparsity over training step.

III. Experiments Evaluation and Discussions

In order to explore the effects of pruning on accuracy and energy consumption in neural networks, we trained and pruned five different networks – LeNet5, LeNet-300-10, LeNet-300-100-10, LeNet-10 [8], and a variation of LeNet5 named here LeNet5-1C – using the MNIST and CIFAR10 datasets.

LeNet5 was the first DNN to achieve commercial success, being used for digits recognition and containing two convolutional layers. LeNet5-1C is similar to LeNet5, but has only one convolutional layer, making it a slightly less complex network. LeNet-300-10 is a simpler NN containing a single hidden layer; nevertheless, it should already posses significant representational power compared to a linear classifier. LeNet-300-100-10 contains one additional hidden layer, which facilitates avoidance of local minima; and finally LeNet-10 does not contain any hidden layers and can be understood as a simple linear classifier – in fact, we name it LeNet-10 here just to unify notation. The non-linear function applied to every layer is the ReLU [4]. Moreover, after each convolutional layer in LeNet5 and LeNet5-1C there is a MaxPool layer.

The MNIST dataset consists of 28×28 grayscale images of handwritten digits (10 classes), 60000 of which are training samples and 10000 are test samples. Figure 5 shows some examples of the images in the MNIST dataset. The LeNet networks were first applied in this dataset [8]. CIFAR10 also has 10 different classes – such as 'bird' and 'airplane' – of 32×32 colour images (3 components each), divided between 50000 training samples and 10000 test samples. Figure 6 shows some examples for the classes *frog*, *car*, *deer* and *truck*.

Fig. 5: Examples of MNIST inputs.

Fig. 6: Examples of CIFAR10 inputs.

Tables I and II summarizes the structure for each network. $C(C,M,K)$ means a convolutional layer with C input components, M output components and $K\times K$ 2D kernels while $FC(X,Y)$ means a fully connected layer with X input neurons and Y output neurons. The small differences in shape between CIFAR10 and MNIST versions are consequences of the different input images sizes.

TABLE I: Neural Networks shapes for CIFAR10.

Neural Network	Shape
LeNet5	C(3, 6, 5)-C(6, 16, 5)-FC(400, 120)-FC(120, 84)-FC(84, 10)
LeNet5-1C	C(3, 16, 5)-FC(400, 120)-FC(120, 84)-FC(84, 10)
LeNet-300-10	FC(3072, 300)-FC(300, 10)
LeNet-300-100-10	FC(3072, 300)-FC(300, 100)-FC(100, 10)
LeNet-10	FC(3072, 10)

Each network was trained for 15-20 epochs, using Stochastic Gradient Descent (SGD) and backpropagation [16] with a learning rate of 0.01 and a momentum of 0.9.

A. Performance and Sparsity Analysis

We applied the AGP method to the pre-trained networks described earlier. For every network we performed an AGP

TABLE II: Neural Networks shapes for MNIST.

Neural Network	Shape
LeNet5	C(1, 6, 5)-C(6, 16, 5)-FC(256, 120)-FC(120, 84)-FC(84, 10)
LeNet5-1C	C(1, 16, 5)-FC(256, 120)-FC(120, 84)-FC(84, 10)
LeNet-300-10	FC(784, 300)-FC(300, 10)
LeNet-300-100-10	FC(784, 300)-FC(300, 100)-FC(100, 10)
LeNet-10	FC(784, 10)

pruning step after each training step during 15 training steps, reaching a final sparsity level of 0.999 for each NN. After each step, we evaluated the network using the test set to find its accuracy at each sparsity level. The accuracy using the MNIST and CIFAR10 datasets are shown in Figures 7 and 8 respectively. Specifically, we show that top-one accuracy, which is the conventional accuracy, where the model prediction must correspond exactly to the correct class.

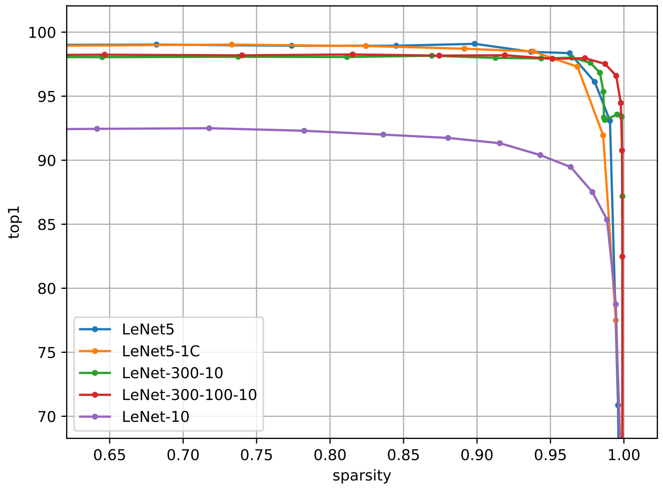

Fig. 7: Relation between sparsity and performance using the MNIST dataset.

As expected, the convolutional networks achieved higher accuracy levels in all datasets, since they have a more complex structure. Both of them presented similar results, although with CIFAR10 the LeNet5-1C performed slightly better. LeNet-300-100-10 however did not show much improvement over LeNet-300-10, even though it possess one more hidden layer. LeNet-10 finished far behind, as expected since it has a much lower representational power by acting as a linear classifier.

More importantly, we note how accuracy could be maintained while reaching a global sparsity of more than 95% for all networks. Simply removing the smallest weights would not achieve the same results; the training steps between each pruning step are essential in order to allow the network to recover from the losses. Moreover, every network could in fact be improved by some level of sparsity. Table III compares the best accuracy obtained through AGP with the original accuracy of each network.

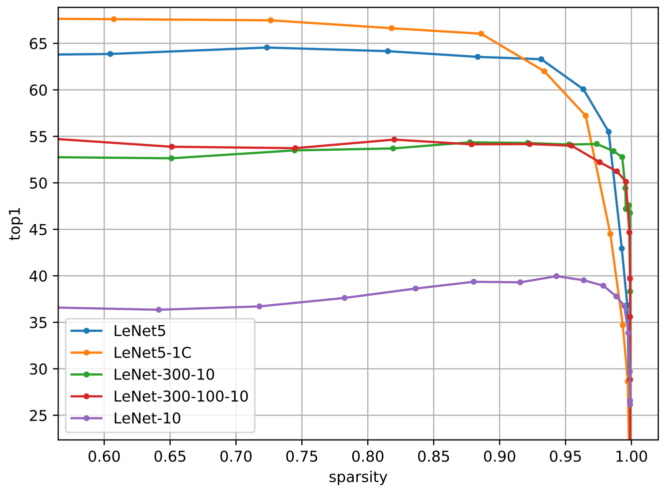

Fig. 8: Relation between sparsity and performance using the CIFAR10 dataset.

TABLE III: Comparison between original and sparse NNs.

Neural Network	MNIST			CIFAR10		
	Dense	Best Sparse		Dense	Best Sparse	
	Acc.	Sparsity	Acc	Acc.	Sparsity	Acc.
LeNet5	98.9	0.898	99.1	62.5	0.724	64.5
LeNet5-1C	98.9	0.733	99.0	67.1	0.457	67.7
LeNet-300-10	98.1	0.869	98.2	48.9	0.878	54.4
LeNet-300-100-10	97.9	0.815	98.3	52.1	0.820	54.6
LeNet-10	92.3	0.718	92.5	35.3	0.943	39.9

The accuracy gains with the MNIST dataset are marginal for all networks. This is reasonable, since the performance for this dataset is already very high. Accuracy with CIFAR10 however was greatly improved; LeNet-300-10 for instance accomplished a performance approximately 5.5% higher. These gains can be understood as a consequence of less overfitting. Since the original networks have much more weights than are necessary – as is shown by their tolerance to pruning – they become prone to memorizing the training set, which diminishes their ability to generalize well. When we remove unnecessary weights, we force the network to learn more general features and therefore reduce the chance of overfitting.

Furthermore, noticing how the pruned networks needs fewer weights than their original versions, one might ask if we could simply use smaller analogous networks for the same datasets and achieve similar results. In order to answer this, we also trained a simplified version of the LeNet-300-10 containing the same number of weights as the sparse versions with accuracies of 97.6% and 53.4% with MNIST and CIFAR10. For the MNIST dataset, there are $28 \times 28 = 784$ input neurons and 300 hundred hidden neurons, accounting for a total of 235200 different weights in its first layer. Through pruning we can reduce this number to approximately 2432 non-zero weights while maintaining accuracy. A fully connected layer with the same number of weights and the same number of input neurons would need to have only 4 neurons in its hidden layer. An analogous calculation shows the same number of hidden neurons for the CIFAR10 dataset. This 'LeNet-4-10' thus contains the same number of non-zero parameters as the sparse LeNet-300-10, but it performs much worse. It achieved a accuracy of only 25% and 68% with the CIFAR10 and MNIST datasets, while the sparse network achieved 53.4% and 97.6% respectively.

As suggested in [7], pruning allows the network to learn not only the weights, but also in a certain sense the relevant connections and therefore the structure of the layers. This means not only that fewer weights are needed, but also that new layer structures are learned. In other words, pruning allows for weights arrangements that would otherwise not be possible by considering only fully connected or convolutional layers with the same sizes. This explains the differences in accuracy found before; although the 'LeNet-4-10' has approximately the same number of weights as the sparse LeNet-300-10, it cannot reproduce the same connections.

B. Energy Consumption Analysis

Different architectures for performing the network computations would achieve different results concerning energy consumption. In order to estimate the reduction from pruning, we consider a simple architecture and dataflow similar to the Non-Local Reuse described in [17], but without inter-array reuse, i.e. the parameters for the multiply-and-accumulate (MAC) operations are always obtained from and saved in a global buffer. We also consider an energy model from [17], based on experimental results using a 65 nm technology, where each MAC operation consumes a normalized energy of 1 and each memory access to the global buffer consumes 6 times it.

Assuming the described model, we can estimate the trade-off between energy consumption and accuracy through counting the total number of MAC operations. From equations (2) and (3) we see that the number of MACs per convolutional layer is given by

$$\#MAC = n \times (H - K + U) \times (W - K + U)/U^2 \tag{5}$$

where $n = C \times K^2 \times M$ is the number of weights in the layer and H, K, U, C and M are as defined in Section II. We note also that a fully connected layer can be computed as a convolution by setting $W = H = K$, $U = 1$ and n to the number of weights in the layer. Furthermore, the number of MAC operations needed in a sparse layer network will be given by $\#MACSparse = (1 - sparsity) \times \#MAC$ since there is no need to compute a MAC where the weight is equal to 0. Finally, the normalized energy consumption can be derived by considering that each MAC operation needs 4 memory accesses: 3 for reading and 1 for writing. This is important to note also that this is a very pessimistic model in terms of parameters reusing and memory accesses. Most architectures today would have different memory levels, such that the most accessed parameters could be obtained with less energy consumption. Moreover, this is also possible to design dataflows in order to increase the possibility of parameters reuse [4], which can be used with a hierarchy of memories in order to further decrease energy consumption. Nevertheless, the model here serves well for a first analysis.

Tables IV and V summarize the results for energy consumption and accuracy with the MNIST and CIFAR10 datasets, respectively. The networks shown in the *sparse* column were chosen such that their accuracies were between 1% and 3% below that of the original networks.

Energy consumption was significantly reduced for all networks, following the expectancy set by the high sparsity levels. LeNet-300-10, LeNet-300-100-10 and LeNet-10 consumed less than 0.5% the original values with the CIFAR10 dataset while LeNet5 and LeNet5-1C used approximately 13.7% and 27.3%. With MNIST, LeNet5, LeNet5-1C, LeNet-300-10, LeNet-300-100-10 and LeNet-10 consumed respectively 5.7%, 14.4%, 1.6%, 0.5% and 5.6% their original values. In general we note that the reduction was smaller with the convolutional networks, since the convolutional layer structure is such that there is already less parameter redundancy. Their accuracies accuracy with CIFAR10 however remained much higher than the other networks. Nonetheless it might be justified to use a different network than a convolutional one with the MNIST dataset if energy is a big constraint, since the other NNs achieved a similar accuracy while consuming much less energy.

TABLE IV: Results for accuracy and relative energy[1] for each neural network architecture with the MNIST dataset.

NN Architecture	Original		Best Sparse		Sparse	
	Acc. [%]	Energy [$\times 10^6$]	Acc. [%]	Energy [$\times 10^6$]	Acc. [%]	Energy [$\times 10^6$]
LeNet5	98.9	7.04	99.1	1.23	96.1	0.402
LeNet5-1C	98.9	6.80	99.0	2.87	97.3	0.982
LeNet-300-10	98.1	5.95	98.2	0.778	96.8	0.0976
LeNet-300-100-10	97.9	6.65	98.3	1.23	96.6	0.0349
LeNet-10	92.3	0.196	92.5	0.0553	90.4	0.0111

[1] Relative energy normalized with one MAC operation (Eq. 5).

TABLE V: Results for accuracy and relative energy[1] for each neural network architecture with the CIFAR10 dataset.

NN Architecture	Original		Best Sparse		Sparse	
	Acc. [%]	Energy [$\times 10^6$]	Acc. [%]	Energy [$\times 10^6$]	Acc. [%]	Energy [$\times 10^6$]
LeNet5	62.5	16.2	64.5	6.77	60.1	2.22
LeNet5-1C	67.1	24.9	67.7	16.5	66.04	6.80
LeNet-300-10	48.9	23.1	54.4	2.83	46.8	0.0258
LeNet-300-100-10	52.1	23.8	54.6	4.28	50.1	0.0996
LeNet-10	35.3	0.768	39.9	0.0437	33.9	0.00177

[1] Relative energy normalized with one MAC operation (Eq. 5).

IV. Conclusion

This paper investigates pruning effects on both accuracy and energy consumption of neural networks. Specifically, we compared five different NNs architectures using two datasets, the MNIST and CIFAR10. Following similar results in the area, we showed that very high sparsity can be achieved for all networks while maintaining accuracy unchanged – results that cannot be obtained with smaller analogous NNs. Moreover, we used a model for the computation of the networks to show that these sparsity levels lead to a significant reduction in energy consumption. We note also that different energy cuts are expected depending on the architecture and dataflow used, though the general trend should remain.

In further research, we plan to explore additional model architectures by considering more complex memory hierarchies and dataflows. Moreover, we consider that it is important to verify whether similar results could still be obtained with more recent datasets and networks, which we intend to verify in further works. Finally, this would also be interesting to consider whether pruning has distinct effects depending on the category of the input or on the metric used. In order evaluate this, we plan to experiment using different metrics for evaluating the class specific results other than accuracy, such as the F-Measure.

Acknowledgment

The authors would like to thank IFRS and CNPq, Capes and Fapergs Brazilian agencies for financial support to our research.

References

[1] J. Misra and I. Saha, “Artificial neural networks in hardware: A survey of two decades of progress,” *Neurocomputing*, vol. 74, no. 1, pp. 239 – 255, 2010, artificial Brains. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S092523121000216X

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[3] L. Deng, J. Li, J.-T. Huang, K. Yao, D. Yu, F. Seide, M. L. Seltzer, G. Zweig, X. He, J. D. Williams *et al.*, “Recent advances in deep learning for speech research at Microsoft.” in *ICASSP*, vol. 26, 2013, p. 64.

[4] V. Sze, Y. Chen, T. Yang, and J. S. Emer, “Efficient Processing of Deep Neural Networks: A Tutorial and Survey,” *CoRR*, vol. abs/1703.09039, 2017. [Online]. Available: http://arxiv.org/abs/1703.09039

[5] S. Han, H. Mao, and W. J. Dally, “Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding,” pp. 1–14, 2015. [Online]. Available: http://arxiv.org/abs/1510.00149

[6] Y. LeCun, J. S. Denker, and S. A. Solla, “Optimal brain damage,” in *Advances in neural information processing systems*, 1990, pp. 598–605.

[7] S. Han, J. Pool, J. Tran, and W. J. Dally, “Learning both Weights and Connections for Efficient Neural Networks,” *CoRR*, vol. abs/1506.02626, 2015. [Online]. Available: http://arxiv.org/abs/1506.02626

[8] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[9] M. Zhu and S. Gupta, “To prune, or not to prune: exploring the efficacy of pruning for model compression,” *arXiv preprint arXiv:1710.01878*, 2017.

[10] Y. LeCun, Y. Bengio *et al.*, “Convolutional networks for images, speech, and time series,” *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.

[11] D. Cornelisse, “An intuitive guide to Convolutional Neural Networks,” https://medium.freecodecamp.org/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050, 2018, accessed: 2018-12-01.

[12] R. Reed, “Pruning algorithms-a survey,” *IEEE Transactions on Neural Networks*, vol. 4, no. 5, pp. 740–747, Sep. 1993.

[13] B. Hassibi and D. G. Stork, “Second order derivatives for network pruning: Optimal brain surgeon,” in *Advances in neural information processing systems*, 1993, pp. 164–171.

[14] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, “Pruning Filters for Efficient ConvNets,” *CoRR*, vol. abs/1608.08710, 2016. [Online]. Available: http://arxiv.org/abs/1608.08710

[15] T. Yang, Y. Chen, and V. Sze, “Designing Energy-Efficient Convolutional Neural Networks using Energy-Aware Pruning,” *CoRR*, vol. abs/1611.05128, 2016. [Online]. Available: http://arxiv.org/abs/1611.05128

[16] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning internal representations by error propagation,” California Univ San Diego La Jolla Inst for Cognitive Science, Tech. Rep., 1985.

[17] Y. Chen, J. Emer, and V. Sze, “Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks,” in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, June 2016, pp. 367–379.

Thomas Fontanari (M’15) is currently an undergraduate final year student of the five-year Computer Engineering degree from the Federal University of Rio Grande do Sul (UFRGS), Porto Alegre, Brazil. His research interests include Digital Signal Processing, Low-power Hardware Architectures, Arithmetic Operators, Approximate Computing, and Machine Learning.

Leandro Mateus Giacominni Rocha (S’16) received a five-year engineering degree in Computer Engineering from the Federal University of Rio Grande do Sul, Porto Alegre, Brazil, in 2016. He also obtained an engineer degree in Electronic Integrated Systems from the Grenoble Institute of Technology, France, in 2015 as part of a double degree program. He is a Ph.D. candidate at the Federal University of Rio Grande do Sul, Porto Alegre, Brazil. Currently, he is a visiting PhD student at KUL and imec working on neural network hardware accelerators for portable devices. His research interests are low-power VLSI architectures, Approximate Computing, Neural Networks Accelerators, Arithmetic Operators, Digital Signal Processing, Machine Learning, Multipliers.

Gustavo Madeira Santana (S’19) is currently an undergraduate final year student of the five-year Computer Engineering degree from the Federal University of Rio Grande do Sul (UFRGS), Porto Alegre, Brazil. His research interests include Video Coding Algorithms, Digital Signal Processing, Low-power Hardware Architectures, Arithmetic Operators, Approximate Computing, and Machine Learning.

Guilherme Paim (S’15) received the five-year engineering degree in Electronics Engineering from the Federal University of Pelotas, Pelotas, Brazil, in 2015. He is Ph.D. candidate at the Federal University of Rio Grande do Sul, Porto Alegre, Brazil. Currently, he is with Karlsruhe Institute of Technology (KIT) as visiting researcher with a Ph.D. sandwich scholarship. His research interests are: Approximate Computing, Low-power VLSI architectures, Arithmetic Operators, Video Coding, Approximate Digital Signal Processing, Mixed-signal circuits, Neural Networks Accelerators, Side channel attack-resistant circuits for Cryptography.

Eduardo Antonio Ceśar da Costa (M’01) received the five-year engineering degree in Electrical Engineering from the University of Pernambuco, Recife, Brazil, in 1988, the M.Sc. degree in electrical engineering from the Federal University of Paraíba, Campina Grande, Paraíba, Brazil, in 1991, and the Ph.D. degree in computer science from the Federal University of Rio Grande do Sul, Porto Alegre, Brazil, in 2002. Part of his doctoral work was developed at the Instituto de Engenharia de Sistemas Computadores (INESC-ID), Lisbon, Portugal. He is currently a full professor at the Catholic University of Pelotas (UCPel), Pelotas, Brazil. He is co-founder and coordinator of the Graduate Program on Electronic Engineering and Computing at UCPel. His research interests are VLSI architectures and low-power design.

Sergio Bampi (M’86-SM’17) received the Electronics Engineer and B.Sc. Physics degrees from Federal University of Rio Grande do Sul (1979). He received the MSEE and Ph.D. in Electrical Engineering degrees from Stanford University in 1982 and 1986, respectively. He is a full professor at the Informatics Institute at the Federal University of Rio Grande do Sul, Brazil, which he joined in 1981. He was a former president of the Brazilian Microelectronics Society, of the FAPERGS Brazilian research funding agency, and CEITEC Technical Director. He is a senior member of IEEE and was a distinguished lecturer of IEEE CAS Society (2009-2010). He has published more than 360 research papers in the fields of CMOS Analog, Digital and RF Design, Video Coding algorithms and hardware architectures, and MOS devices. He was Technical Program Chair of SBCCI (1997, 2005), IEEE LASCAS (2013), SBMICRO Congress (1989), and served on TPC Committees of ICCAD, ICCD, SBCCI, ICM, LASCAS, VLSI-SoC and many other international conferences.