SCANN: Synthesis of Compact and Accurate Neural Networks

Shayan Hassantabar, Zeyu Wang, and Niraj K. Jha, *Fellow, IEEE*

Abstract—Artificial neural networks (ANNs) have become the driving force behind recent artificial intelligence (AI) research. With the help of a vast amount of training data, neural networks can perform better than traditional machine learning algorithms in many applications, such as image recognition, speech recognition, and natural language processing. An important problem with implementing a neural network is the design of its architecture. Typically, such an architecture is obtained manually by exploring its hyperparameter space and kept fixed during training. The architecture that is selected is the one that performs the best on a hold-out validation set. This approach is both time-consuming and inefficient as it is in essence a trial-and-error process. Another issue is that modern neural networks often contain millions of parameters, whereas many applications require small inference models due to imposed resource constraints, such as energy constraints on battery-operated devices. Also, whereas ANNs have found great success in big-data applications, there is also significant interest in using ANNs for medium- and small-data applications that can be run on energy-constrained edge devices. However, efforts to migrate ANNs to such devices typically entail a significant loss of classification accuracy. To address these challenges, we propose a neural network synthesis methodology, called SCANN, that can generate very compact neural networks without loss in accuracy for small and medium-size datasets. With the help of three basic operations, connection growth, neuron growth, and connection pruning, SCANN synthesizes an arbitrary feed-forward neural network. These neural networks do not necessarily have a multilayer perceptron structure. SCANN encapsulates three synthesis methodologies that apply a repeated grow-and-prune paradigm to three architectural starting points. We also use dimensionality reduction methods to reduce the feature size of the datasets, so as to alleviate the curse of dimensionality. Our final synthesis methodology consists of three steps: dataset dimensionality reduction, neural network compression in each layer, and neural network compression with SCANN. We demonstrate the efficacy of this approach on the medium-size MNIST dataset by comparing our synthesized neural networks to the well-known LeNet-5 baseline. Without any loss in accuracy, SCANN generates a 46.3$\times$ smaller network than the LeNet-5 Caffe model. We also evaluate the efficiency of using dimensionality reduction alongside SCANN on nine small to medium-size datasets. Using this methodology enables us to reduce the number of connections in the network by up to $5078.7\times$ (geometric mean: 82.1$\times$), with little to no drop in accuracy. We also show that our synthesis methodology yields neural networks that are much better at navigating the accuracy vs. energy efficiency space. This would enable neural network based inference even for IoT sensors.

Index Terms—Architecture synthesis; compact network; compression; dimensionality reduction; energy efficiency; neural network.

1 Introduction

Artificial neural networks (ANNs) have a long history, dating back to 1950's [1]. However, interest in ANNs has waxed and waned over the years. The recent spurt in interest in ANNs is due to large datasets becoming available, enabling ANNs to be trained to high accuracy. This trend is also due to a significant increase in compute power that speeds up the training process. ANNs demonstrate very high classification accuracies for many applications of interest, e.g., image recognition [2], speech recognition [3], and machine translation [4]. ANNs have also become deeper, with tens to hundreds of layers. Thus, the phrase 'deep learning' is often associated with such neural networks [5]. Deep learning refers to the ability of ANNs to learn hierarchically, with complex features built upon simple ones.

An important challenge in deploying ANNs in practice is their architecture design, since the ANN architecture directly influences the learnt representations and thus the performance. Typically, it takes researchers a huge amount of time through much trial-and-error to find a good architecture because the search space is exponentially large with respect to many of its hyperparameters. As an example, let us consider a convolutional neural network (CNN) often used in image recognition tasks. Its various hyperparameters, such as depth, number of filters in each layer, kernel size, how feature maps are connected, etc., need to be determined when designing an architecture. Improvements in such architectures often take several years of effort, as evidenced by the evolution of various architectures for the ImageNet dataset: AlexNet [2], GoogleNet [6], [7], ResNet [8]–[10], and DenseNet [11].

Another challenge ANNs pose is that to obtain their high accuracy, they need to be designed with a large number of parameters. This negatively impacts both the training and inference times. For example, modern deep CNNs often have millions of parameters and take days to train even with powerful graphics processing units (GPUs). However, making the ANN models compact and energy-efficient may enable them to be moved from the cloud to the edge, leading to benefits in communication energy, network bandwidth, and security. The challenge is to do so without degrading accuracy. This is currently a very active area of research [12]–[24].

As the number of features or dimensions of the dataset increases, in order to generalize accurately, we need exponentially more data. This is another challenge which is referred to as the *curse of dimensionality*. Hence, one way to reduce the

The first two authors contributed equally.
This work was supported by IP Group and NSF Grant No. CNS-1617640.
Shayan Hassantabar, Zeyu Wang and Niraj K. Jha are with the Department of Electrical Engineering, Princeton University, Princeton, NJ, 08544 USA, e-mail:{seyedh, zeyuwang,jha}@princeton.edu.

need for large amounts of data is to reduce the dimensionality of the dataset. In addition, with the same amount of data, by reducing the number of features, the accuracy of the inference model may also improve to a degree. However, beyond a certain point, which is dataset-dependent, reducing the number of features may lead to loss of information, which may lead to inferior classification results.

In this paper, we address the aforementioned challenges by introducing an ANN synthesis methodology, called SCANN. The ANNs it synthesizes are not necessarily multilayer perceptrons (MLPs). SCANN allows skipped connections, instead of enforcing a layer-by-layer connection structure, in order to address the limitation of conventional ANNs that fixes their depth while training. SCANN uses three basic operations: connection growth, neuron growth, and connection pruning. It generates a feed-forward architecture with arbitrary depth. It uses three effective training schemes that enable it to generate ANNs that are much smaller in size and similar in performance relative to prior ANNs.

Moreover, in order to alleviate the curse of dimensionality, we use dimensionality reduction methods to reduce the feature size of the datasets, before using the SCANN methodology to generate compact neural networks. This methodology consists of three basic blocks: dimensionality reduction, neural network compression in each layer, followed by another neural network compression step with SCANN. We refer to this methodology as *DR+SCANN*.

For the LeNet-5 Caffe model [23], [25] derived for the MNIST [23] dataset, SCANN generates a network with only 9.3K parameters (representing a 46.3× compression ratio relative to the baseline), yet providing similar performance as the baseline. To further show the efficacy of SCANN, we experiment with several small to medium-size datasets. Although on two datasets, SCANN achieves a compression ratio of $1.1\times$ and $1.5\times$ while losing less than 3% in accuracy, on the other seven, SCANN generates ANNs that are $1.5\times$ to $317.4\times$ smaller than the baseline architecture, without any drop in classification accuracy. In addition, by combining dimensionality reduction with SCANN (i.e., DR+SCANN), we demonstrate a compression ratio between $1.2\times$ to $5078.7\times$, with little to no drop in model performance on these datasets. These results show that our final synthesis framework (DR+SCANN) has an even higher compression power than SCANN, and can significantly reduce the number of connections in the network, without degrading performance. Moreover, we demonstrate that DR+SCANN yields ANNs that are very energy-efficient, yet offering similar accuracy compared to other methods. This opens the door for such ANNs to be used in IoT sensors.

The paper is organized as follows. Section 2 describes related work. Section 3 describes the SCANN synthesis methodology in detail. Section 4 explains the methodology that combines dimensionality reduction with SCANN. Section 5 provides results of synthesis on various benchmarks. Section 6 provides a short discussion. Finally, Section 7 concludes the paper.

2 Related Work

In this section, we review some of the previous work in two related areas: dimensionality reduction and automatic architecture synthesis.

2.1 Dimensionality Reduction

The high dimensionality of many datasets used in various applications of machine learning leads to the curse of dimensionality problem. Therefore, researchers have explored dimensionality reduction methods to improve the performance of machine learning models by decreasing the number of features. Traditional dimensionality reduction methods include Principal Component Analysis (PCA), Kernel PCA, Factor Analysis (FA), Independent Component Analysis (ICA), as well as Spectral Embedding methods. Some graph-based methods include Isomap [26] and Maximum Variance Unfolding [27]. FeatureNet [28] uses community detection in small sample size datasets to map high-dimensional data to lower dimensions. Other dimensionality reduction methods include stochastic proximity embedding (SPE) [29], Linear Discriminant Analysis (LDA), and t-distributed Stochastic Neighbor Embedding (t-SNE) [30]. A detailed survey of dimensionality reduction methods can be found in [31].

2.2 Automatic Architecture Synthesis

There are three different categories of automatic architecture synthesis methods that have been proposed by researchers: evolutionary algorithm, reinforcement learning algorithm, and structure adaptation algorithm.

2.2.1 Evolutionary Algorithm

The use of an evolutionary algorithm to select an ANN architecture dates back to 1989 [32]. One of the seminal works in neuroevolution is the NEAT algorithm [33], which uses direct encoding of every neuron and connection to simultaneously evolve the network architecture and weights through weight mutation, connection mutation, node mutation, and crossover. Recent years have seen extensions of the evolutionary algorithm to generate CNNs. For example, Xie and Yuille [34] use a concise binary representation of network connections, and demonstrate a comparable classification accuracy to previous human-designed architectures.

2.2.2 Reinforcement Learning Algorithm

A recent trend in automatic ANN architecture synthesis is to use reinforcement learning. Zoph and Le [35] use a recurrent neural network as a controller to generate a string that specifies the network architecture. They use the performance of the generated network on a validation dataset as the reward signal to compute the policy gradient and update the controller. In a later work [36], the authors define a different search space and use the controller to obtain a building block instead of the whole network. They showed that the convolutional cells obtained by learning performed on the CIFAR-10 dataset can be successfully transferred to architectures for other datasets. They achieve a state-of-the-art classification accuracy on ImageNet.

2.2.3 Structure Adaptation Algorithm

Several previous works achieve compact and accurate neural networks through structure adaptation algorithms. One such method is network pruning, which has been used in several

works [12], [37]–[42]. Structure adaptation algorithms can be constructive or destructive. Constructive algorithms start from a small neural network and grow it into a larger more accurate neural network. Destructive algorithms start from a large neural network and prune connections and neurons to get rid of the redundancy while maintaining accuracy. NeST [37] is a network synthesis tool that combines both the constructive and destructive approaches in a grow-and-prune synthesis paradigm. It is used to synthesize compact and accurate architectures for the MNIST and ImageNet datasets. However, its limitation is that growth and pruning are both performed at a specific ANN layer. Thus, network depth cannot be adjusted and is fixed throughout training. In the next section, we will show this problem can be solved by synthesizing a general feed-forward network instead of an MLP architecture, allowing the ANN depth to be changed dynamically during training.

Several works have also proposed more efficient building blocks for CNN architectures [18], [20], [21], [24], [43]–[45]. They result in compact networks, with much fewer parameters, while maintaining or improving performance. Platform-aware search for an optimized NN architecture has also been used in this area. Yin et al. [46] combine the grow-prune synthesis methodology with hardware-guided training to achieve compact long short-term memory (LSTM) cells. The authors of [19] train an ANN to satisfy pre-defined resource constraints, such as latency and energy consumption, with the help of a pre-generated accuracy predictor.

Orthogonal to the above works, quantization has also been used to reduce computations in a network with little to no accuracy drop [13], [17], [47].

3 SCANN Synthesis Methodology

In this section, we first propose a technique to address the limitation of prior work that requires the ANN depth to be fixed. Then we introduce three basic architecture-changing techniques that enable the synthesis of an optimized feed-forward network architecture. Finally, we describe three training schemes that can be used to synthesize the network architecture.

3.1 Depth Change

To address the problem of having to fix the ANN depth during training in prior work, we adopt a general feed-forward architecture instead of an MLP structure. Specifically, a hidden neuron can receive inputs from any neuron activated before it (including input neurons), and can feed its output to any neuron activated after it (including output neurons). In this setting, depth is determined by how hidden neurons are connected and thus can be changed through rewiring of hidden neurons. As shown in Fig. 1, depending on how the hidden neurons are connected, they can form one, two, or three hidden layers.

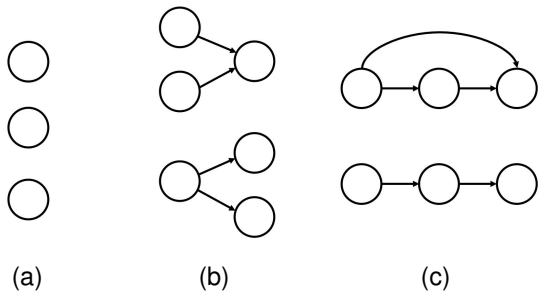

Fig. 1. Connection pattern determines network depth. Only hidden neurons are shown. (a) One hidden layer, (b) two hidden layers, and (c) three hidden layers.

3.2 Overall Workflow

The overall workflow for architecture synthesis is shown in Algorithm 1, the synthesis process iteratively alternates between architecture change and weight training. Thus, the network architecture evolves along the way. After a specified number of iterations, the checkpoint that achieves the best performance on the validation set is output as the final network. Next, we first elaborate on the three basic architecture-changing operations, and then introduce three different training schemes based on how the architectures evolve.

Algorithm 1 Automatic architecture synthesis

Input: Initial network architecture A_{init}, weights W_{init}, and maximum number of iterations I_{max}
  while maximum iterations I_{max} not reached **do**
    (a) Perform one of the three basic architecture-changing operations
    (b) Train weights of the network and test its performance on the validation set
  end while
Output: Final network architecture A_{final} and associated weights W_{final} that achieve the best performance on the validation set

3.3 Basic Architecture-changing Operations

Three basic operations, connection growth, neuron growth, and connection pruning, are used to adjust the network architecture, in order to evolve a feed-forward network just through these operations. Fig. 2 shows a simple example in which an MLP architecture with one hidden layer evolves into a non-MLP architecture with two hidden layers with a sequence of basic operations mentioned above.

Next, we describe these three operations. We denote the ith hidden neuron as n_i, its activity as x_i, and its preactivity as u_i, where $x_i = f(u_i)$ and f is the activation function. We denote the depth of n_i by D_i and the loss function by L. Finally, we denote the connection between n_i and n_j, where $D_i < D_j$, as w_{ij}. In our implementation, we use masks to mask out the pruned weights.

3.3.1 Connection Growth

Connection growth adds connections between neurons that are unconnected. The initial weights of all newly added connections are set to 0. Depending on how connections can be added, we use three different methods, as shown in Algorithm 2.

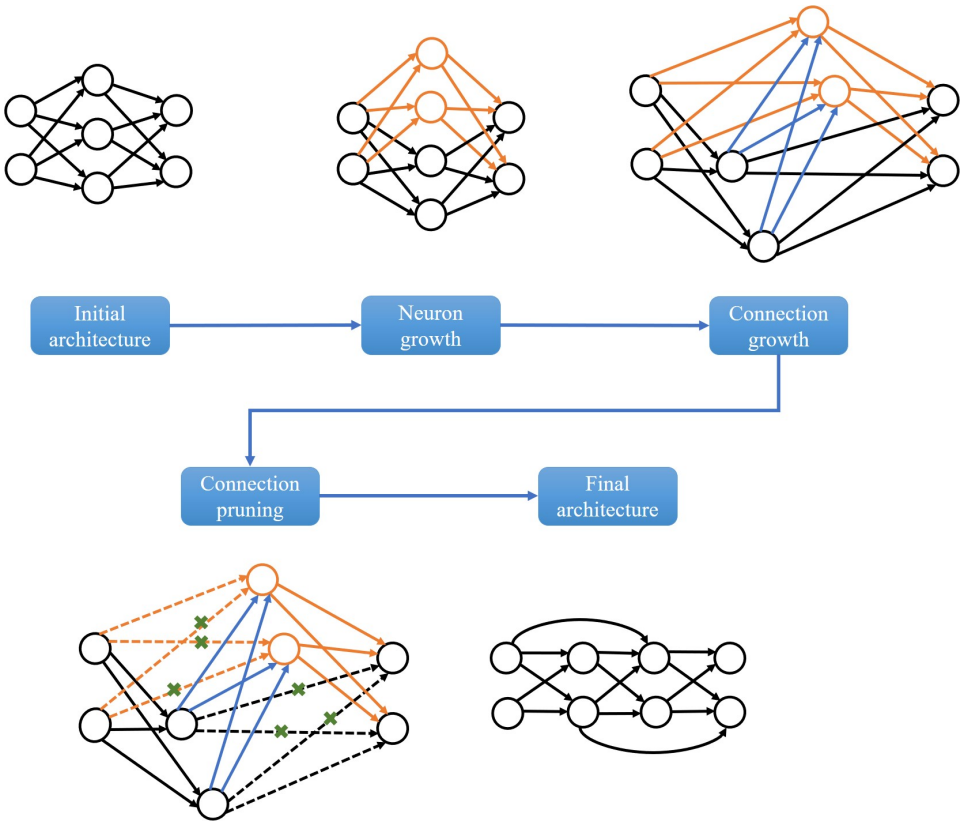

Fig. 2. An MLP architecture with one hidden layer evolves into a non-MLP architecture with two hidden layers through a sequence of neuron growth, connection growth, and connection pruning.

- **Gradient-based growth:** Gradient-based growth was proposed by Dai et al. [37]. It adds connections that tend to reduce the loss function L significantly. Suppose two neurons n_i and n_j are not connected and $D_i \leq D_j$, then gradient-based growth adds a new connection w_{ij} if $\left|\frac{\partial L}{\partial u_j} x_i\right|$ is large.
- **Full growth:** Full growth restores all possible connections to the network.
- **Random growth:** Random growth randomly picks some inactive connections and adds them to the network.

Algorithm 2 Connection growth algorithm

Input: Network N, weight matrix W, mask matrix C, data batch D, threshold t
  if full growth **then**
    set all elements in C to 1
  else if random growth **then**
    randomly set some elements in C to 1
  else if gradient-base growth **then**
    forward propagation through N using data D and then back propagation
    compute $g_{ij} = \left|\frac{\partial L}{\partial u_j} x_i\right|$
    For $g_{ij} > t$, set $c_{ij} = 1, w_{ij} = 0$
  end if
Output: Modified weight matrix W and mask matrix C

3.3.2 Neuron Growth

Neuron growth adds new neurons to the network, thus increasing network size over time. There are two possible methods for doing this, as shown in Algorithm 3. First, drawing an analogy from biological cell division, neuron growth can be achieved by duplicating an existing neuron. To break the symmetry, random noise is added to the weights of all the connections related to this newly added neuron. The specific neuron that is duplicated can be selected in two ways:

- **Activation-based selection:** Activation-based selection selects neurons with a large activation for duplication.
- **Random selection:** Random selection randomly selects neurons for duplication.

Second, instead of duplicating existing neurons, new neurons with random initial weights and random initial connections with other neurons may be added to the network.

3.3.3 Connection Pruning

Connection pruning disconnects previously connected neurons and reduces the number of network parameters. If all connections associated with a neuron are pruned, then the neuron is removed from the network. We adopt a widely-used method [12], [13], [37] to prune connections with small magnitude, as shown in Algorithm 4. The rationale behind it is that since small weights have a relatively small influence on the network, ANN performance can be restored through retraining after pruning.

Algorithm 3 Neuron growth algorithm

Input: Network N, weight matrix W, mask matrix C, data batch D, a candidate neuron n_j to be added
  if neuron division **then**
    if activation-based selection **then**
      forward propagation through N using data D
      $i = argmax\ u_i$
    else if random selection **then**
      randomly pick an active neuron n_i
    end if
    $c_{j\cdot} = c_{i\cdot}, c_{\cdot j} = c_{\cdot i}$
    $w_{j\cdot} = w_{i\cdot} + noise, w_{\cdot j} = w_{\cdot i} + noise$
  else if random growth **then**
    randomly set elements of $c_{j\cdot}$ and $c_{\cdot j}$ to 1
    randomly initialize $w_{j\cdot}$ and $w_{\cdot j}$
  end if
Output: Modified weight matrix W and mask matrix C

Algorithm 4 Connection pruning algorithm

Input: Weight matrix W, mask matrix C, threshold t
  for all w_{ij} **do**
    if $|w_{ij}| < t$ **then**
      $c_{ij} = 0$
    end if
  end for
Output: Modified weight matrix W and mask matrix C

3.4 Training Schemes

In practice, depending on how the initial network architecture A_{init} and basic operations in Step (a) of Algorithm 1 are chosen, we adopt three training schemes in our experiments, as explained next.

3.4.1 Scheme A

Scheme A is a constructive approach, where we start with a tiny network, and gradually increase the network size. This can be achieved by performing connection and neuron growth more often than connection pruning or carefully selecting the growth and pruning rates, such that each growth operation grows a large number of connections and neurons, while each pruning operation prunes a small number of connections.

3.4.2 Scheme B

Scheme B is a destructive approach, where we start with an over-parameterized network and end up with a small network. There are two possible ways to accomplish this. First, similar to the technique in [12], [37], we can iteratively prune a small number of network connections and then train the weights. This gradually reduces network size and finally results in a small network after many iterations. Another approach is that, instead of pruning the network gradually, we can aggressively prune the network to a tiny size. However, to make this approach work, we need to repeatedly prune the network and then grow the network back, rather than just perform a one-time pruning. In practice, we find the second approach works better.

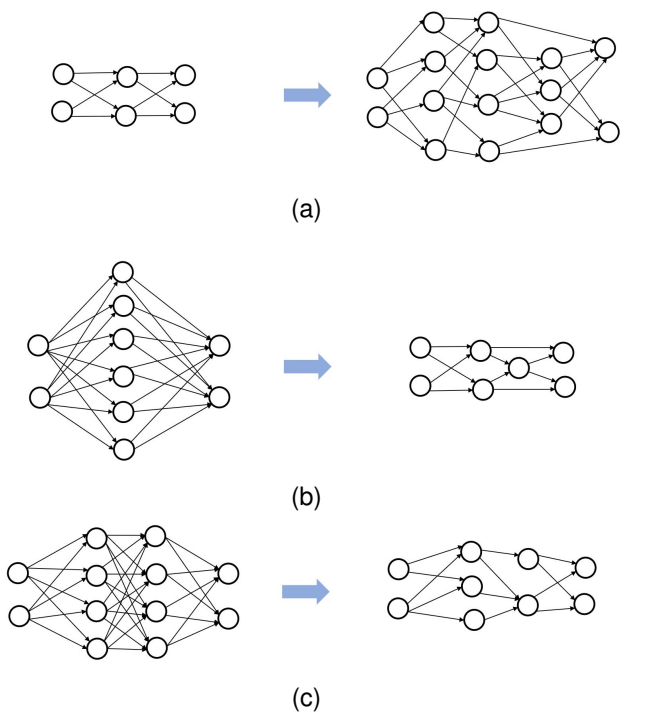

Fig. 3. Illustration of the three training schemes. Shown here are the initial and final architectures: (a) Scheme A, (b) Scheme B, and (c) Scheme C.

3.4.3 Scheme C

Scheme B also works with MLP architectures, with only a small adjustment in connection growth such that only connections between adjacent layers are added and not skipped connections. For clarity, we give another name to MLP-based Scheme B: Scheme C. Note that Scheme C is similar to the iterative hard thresholding technique proposed in [48]. Besides, Scheme C can also be viewed as an iterative version of the dense-sparse-dense technique proposed in [49], with the aim of generating compact networks instead of improving performance of the original architecture.

Fig. 3 shows examples of the initial and final architectures for each scheme. Both Schemes A and B evolve general feed-forward architectures, thus allowing network depth to be changed during training. Scheme C evolves an MLP structure, thus keeping the depth fixed.

4 Dimensionality Reduction + SCANN

In this section, we propose a methodology to synthesize compact neural networks by combining dimensionality reduction (DR) and SCANN, which we refer to as DR+SCANN. Fig. 4 shows the block diagram of the methodology. It begins by obtaining a very accurate baseline architecture by progressively increasing the number of hidden layers. Its other main parts are dataset dimensionality reduction and two neural network compression steps that are discussed next.

4.1 Dataset Modification

Dataset modification entails normalizing the dataset and reducing its dimensionality. All feature values are normalized to the range [0,1]. Reducing the number of features in the

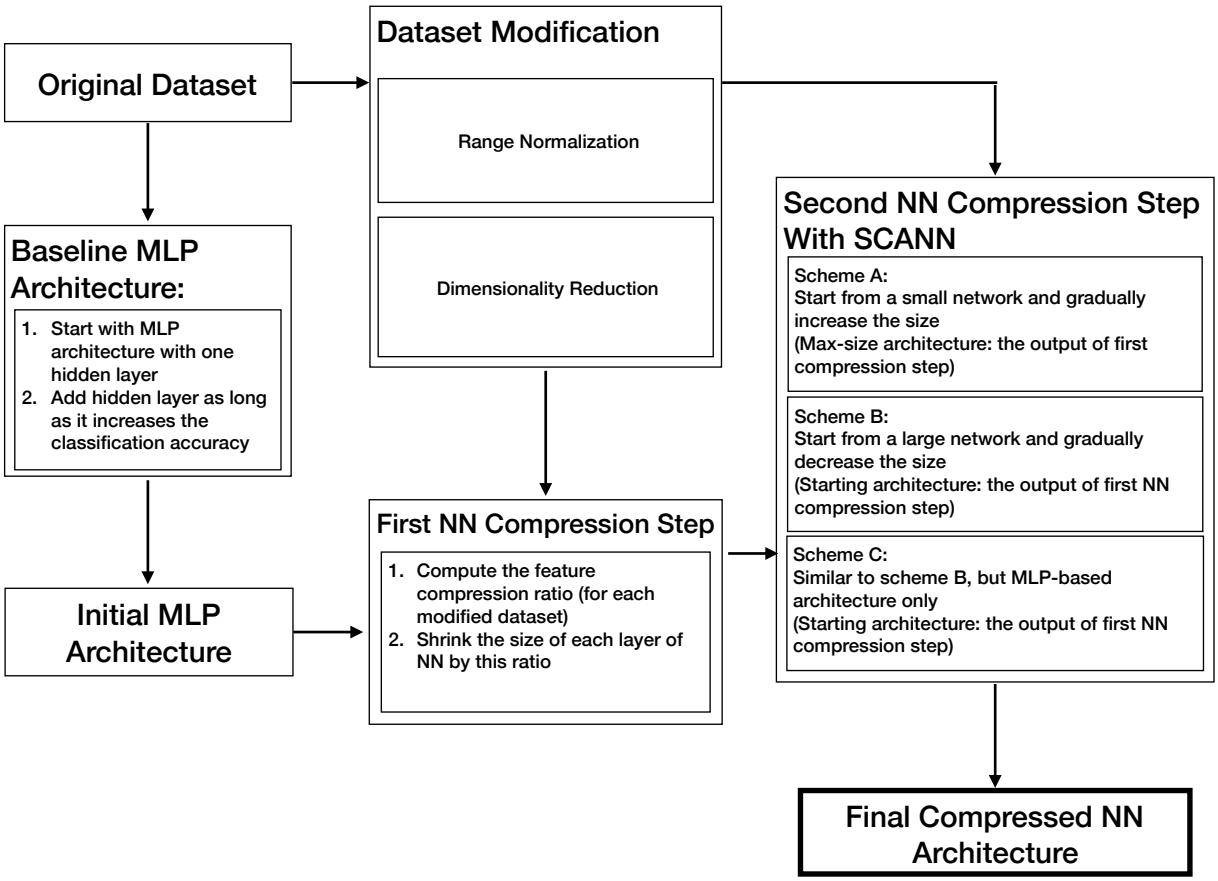

Fig. 4. Block diagram of DR+SCANN: (a) dimensionality reduction into a lower-dimensional space, (b) reduction in the number of neurons in each layer of the initial MLP architecture by the same feature compression ratio, and (c) further compression of the neural network using the three training schemes from SCANN. The process of finding the baseline MLP architecture is also shown.

dataset is aimed at alleviating the effect of the curse of dimensionality and increasing data classifiability. This way, an $N \times d$-dimensional dataset is mapped onto an $N \times k$-dimensional space, $k < d$, using various dimensionality reduction methods. We explore 11 such methods, including four random projection (RP) methods.

RP is used to reduce data dimensionality based on the Johnson-Lindenstrauss lemma [50], [51]. The essence of this lemma is that if the data points are in a space of sufficiently high dimension, they can be projected onto a suitable lower dimension, while approximately maintaining inter-point distances. More precisely, this lemma shows that the distance between the points change only by a factor of $(1 \pm \varepsilon)$, when they are randomly projected onto the subspace of $\mathcal{O}(\log \frac{n}{\varepsilon^2})$ dimensions, for any $0 < \varepsilon < 1$.

The RP matrix Φ can be generated in several ways. Here, we discuss four RP matrices we used. One approach is to generate Φ using a Gaussian distribution. In this case, the entries $\phi_{i,j}$ are i.i.d. samples drawn from a Gaussian distribution $\mathcal{N}(0, \frac{1}{k})$. Another RP matrix can be obtained by sampling entries from $\mathcal{N}(0, 1)$. These entries are shown below.

$$\phi_{ij}^1 \sim \mathcal{N}(0, \frac{1}{k}) \qquad \phi_{ij}^2 \sim \mathcal{N}(0, 1)$$

Achlioptas [52] proposed several other sparse RP matrices. Two of these proposals are as follows, where entries ϕ_{ij}'s are independent random variables that are drawn based on the following probability distributions:

$$\phi_{ij}^3 = \begin{cases} +1 & \text{with probability } \frac{1}{2} \\ -1 & \text{with probability } \frac{1}{2} \end{cases}$$

$$\phi_{ij}^4 = \sqrt{\frac{3}{k}} \begin{cases} 1 & \text{with probability } \frac{1}{6} \\ 0 & \text{with probability } \frac{2}{3} \\ -1 & \text{with probability } \frac{1}{6} \end{cases}$$

The other dimensionality reduction methods that we used include PCA, Polynomial Kernel PCA, Gaussian Kernel PCA, FA, Isomap, ICA, and Spectral Embedding. Implementations of these methods are obtained from the Scikit-learn machine learning library [53].

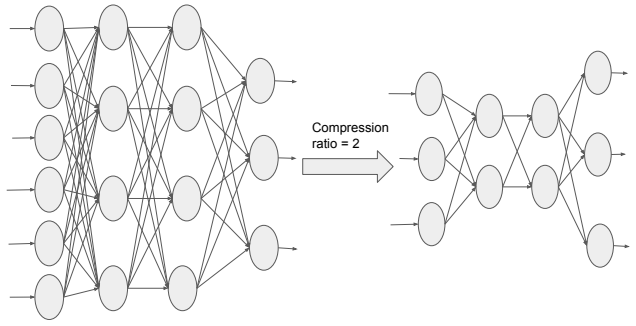

Fig. 5. Compressing the neural network by a compression ratio of 2: the number of neurons in each layer, except the last layer, is reduced by a factor of 2.

4.2 Neural Network Compression in Each Layer

Dimensionality reduction maps the dataset into a vector space of lower dimension. As a result, as the number of features reduces, the number of neurons in the input layer of the neural network decreases accordingly. However, since the dataset dimension is reduced, one might expect the task of classification to become easier. This means we can reduce the number of neurons in all layers, not just the input layer. This step reduces the number of neurons in each layer of the neural network by the *feature compression ratio* in the dimensionality reduction step (see Fig. 4), except for the output layer. Fig. 5 shows an example of this process of compressing neural networks in each layer. We refer to this dimensionality reduction stage as *DR*.

4.3 Neural Network Compression With SCANN

We input several neural network architectures obtained from the output of the first neural network compression step to SCANN. These architectures correspond to the best three classification accuracies, as well as the three most compressed networks that meet the baseline accuracy of the initial MLP architecture, as evaluated on the validation set.

SCANN uses the corresponding reduced-dimension dataset. In Scheme A, we need to set the maximum number of connections in the network. We set this value to the number of connections in the neural network that results from the first compression step. This way, the final neural network will become smaller. Schemes B and C require the maximum number of neurons and the maximum number of connections to be initialized. In addition, in these two training schemes, the final number of connections in the network also needs to be set. Furthermore, the number of layers in the MLP architecture synthesized by Scheme C needs to be predetermined. We initialize these parameters using the network architecture that is output from first neural network compression.

5 Experimental Results

In this section, we evaluate the performance of SCANN and DR+SCANN on several small- to medium-size datasets. Table 5 shows the characteristics of these datasets.

The evaluation results are divided into two parts. Section 5.1 discusses results obtained by SCANN when applied to the widely used MNIST dataset. Compared to related work, SCANN generates neural networks with better classification accuracy and fewer parameters. In Section 5.2, we show results of experiments on nine other datasets. We demonstrate that the ANNs generated by SCANN are very compact and energy-efficient, while maintaining performance. These results open up opportunities to use SCANN-generated ANNs in energy-constrained edge devices and IoT sensors.

5.1 Experiments with MNIST

MNIST is a well-studied dataset of handwritten digits. It contains 60000 training images and 10000 test images. We set aside 10000 images from the training set as the validation set. We adopt the Lenet-5 Caffe model [23], [25] that is widely used in related works [12], [54], [55]. For Schemes A and B, the feed-forward part of the network is learnt by SCANN, whereas the convolutional part is kept the same as in the baseline (Scheme A does not make any changes to the baseline, but Scheme B prunes the connections). For Scheme C, SCANN starts with the baseline architecture, and only learns the connections and weights, without changing the depth of the network. All experiments use the stochastic gradient descent (SGD) optimizer with a learning rate of 0.03, momentum of 0.9, and weight decay of 1e-4. No other regularization technique like dropout or batch normalization is used. We run each experiment five times and report the average performance.

The LeNet-5 Caffe model contains two convolutional layers with 20 and 50 filters, and also one fully-connected hidden layer with 500 neurons. For Scheme A, we start with 400 hidden neurons in the feed-forward part, randomly prune out 95 percent of the connections in the beginning and then iteratively perform a sequence of connection growth that activates 30 percent of all connections and connection pruning that prunes 25 percent of existing connections. For Scheme B, we start with 400 hidden neurons in the feed-forward part and iteratively perform a sequence of connection pruning such that 3.3K connections are left in the convolutional part and 16K connections are left in the feed-forward part, and connection growth such that 90 percent of all connections are restored. For Scheme C, we start with a fully connected baseline architecture and iteratively perform a sequence of connection pruning such that 3.3K connections are left in the convolutional part and 6K connections are left in the feed-forward part, and connection growth such that all connections are restored.

Table 2 summarizes the results. The baseline error rate is 0.72% with 430.5K parameters. The most compressed model generated by SCANN contains only 9.3K parameters (with a compression ratio of 46.3× over the baseline), achieving a 0.72% error rate when using Scheme C. Scheme A obtains the best error rate of 0.68%, however, with a lower compression ratio of 2.3×. For a fair comparison, we implement the method given in [12] on the same data split.

5.2 Experiments with Other Datasets

Though SCANN demonstrates very good compression ratios for LeNets on the medium-size MNIST dataset at similar or

TABLE 1
Characteristics of the datasets

Dataset	Training Set	Validation Set	Test Set	Features	Classes
MNIST	50000	10000	10000	784	10
Sensorless Drive Diagnosis	40509	9000	9000	48	11
Human Activity Recognition (HAR)	5881	1471	2947	561	6
Musk v2	4100	1000	1974	166	2
Pen-Based Recognition of Handwritten Digits	5995	1499	3498	16	10
Landsat Satellite Image	3104	1331	2000	36	6
Letter Recognition	10500	4500	5000	16	26
Epileptic Seizure Recognition	6560	1620	3320	178	2
Smartphone Human Activity Recognition	6121	153	3277	561	12
DNA	1400	600	1186	180	3

TABLE 2
Comparison of different methods on the LeNet-5 Caffe model

Methods	Error rate	Weights	Compression ratio
Baseline	0.72%	430.5K	1.0×
Network pruning [12]	0.77%	34.5K	12.5×
Scheme A	0.68%	184.6K	2.3×
Scheme B	0.72%	19.3K	22.3×
Scheme C	0.72%	9.3K	46.3×

better accuracy, one may ask if SCANN can also generate compact neural networks from other medium and small datasets. To answer this question, we experiment with nine datasets from the UCI machine learning repository [56] and Statlog collection [57]. Next, we present evaluation results on these datasets.

SCANN experiments are based on the Adam optimizer with a learning rate of 0.01 and weight decay of 1e-3. We compare results obtained by DR+SCANN with those obtained by only applying SCANN, and also DR without using SCANN in a secondary compression step. Table 3 shows the classification accuracy obtained. The MLP column shows the accuracy of the MLP baseline for each dataset. For all the other methods, we present two columns, the left of which shows the highest achieved accuracy (H.A.) whereas the right one shows the result for the most compressed network (M.C.). Furthermore, for the DR columns, the dimensionality reduction method employed is shown in parentheses. Table 4 shows the number of parameters in the network for the corresponding columns in Table 3.

SCANN-generated networks show improved accuracy for six of the nine datasets, as compared to the MLP baseline. The accuracy increase is between 0.41% to 9.43%. These results correspond to networks that are $1.2\times$ to $42.4\times$ smaller than the base architecture. Furthermore, DR+SCANN shows improvements on the highest classification accuracy on five out of the nine datasets, as compared to SCANN-generated results.

In addition, SCANN yields ANNs that achieve the baseline accuracy with fewer parameters on seven out of the nine datasets. For these datasets, the results show a connection compression ratio between $1.5\times$ to $317.4\times$. Moreover, as shown in Tables 3 and 4, combining dimensionality reduction with SCANN helps achieve higher compression ratios. For these seven datasets, DR+SCANN can meet the baseline accuracy with a $28.0\times$ to $5078.7\times$ smaller network. This shows a significant improvement over the compression ratio achievable by just using SCANN.

We also report the performance of applying DR without the benefit of the SCANN synthesis step. While these results show improvements, DR+SCANN can be seen to have much more compression power, relative to when DR and SCANN are used separately. This points to a synergy between DR and SCANN.

Although the classification performance is of great importance, in applications where computing resources are limited, e.g., in battery-operated devices, energy efficiency might be one of the most important concerns. Thus, energy performance of the algorithms should also be taken into consideration in such cases. To evaluate the energy performance, we use the energy analysis method proposed in [58], where the energy consumption for inference is calculated based on the number of multiply-accumulate (MAC) and comparison operations and the number of SRAM accesses. For example, a multiplication of two matrices of size $M \times N$ and $N \times K$ would require $(M \cdot N \cdot K)$ MAC operations and $(2 \cdot M \cdot N \cdot K)$ SRAM accesses. In their model, a single MAC operation, SRAM access, and comparison operation implemented in a $130nm$ CMOS process (which may be an appropriate technology for many IoT sensors) consumes 11.8 pJ, 34.6 pJ and 6.16 fJ, respectively. Table 5 shows the energy consumption estimates per inference for the corresponding models discussed in Tables 3 and 4. DR+SCANN can be seen to have the best overall energy performance. Except for the Letter dataset (for which the energy reduction is only 17 percent), the compact ANNs generated by DR+SCANN consume one to four orders of magnitude less energy than the baseline MLP models. Thus, this synthesis methodology is suitable for heavily energy-constrained devices, such as IoT sensors.

6 Discussion

The advantages of SCANN are derived from its core benefit: the network architecture is allowed to dynamically evolve

TABLE 3
Test accuracy comparison

Dataset	MLP	DR (H.A.)	DR (M.C.)	SCANN (H.A.)	SCANN (M.C.)	DR+SCANN (H.A.)	DR+SCANN (M.C.)
SenDrive	93.53%	99.07% (FA)	97.99% (FA)	97.10%	93.63%	99.34%	94.20%
HAR	95.01%	95.04% (ICA)	95.04% (ICA)	95.52%	95.52%	95.28%	95.08%
Musk	98.68%	98.83% (FA)	98.78% (FA)	99.09%	98.83%	98.08%	98.08%
Pendigits	97.22%	97.51% (Isomap)	97.39% (Isomap)	97.22%	97.22%	97.93%	97.65%
SatIm	91.30%	91.10% (PCA)	91.10% (PCA)	90.10%	90.10%	89.40%	89.40%
Letter	95.24%	94.92% (PCA)	94.92% (PCA)	92.60%	92.60%	92.70%	92.70%
Seizure	87.53%	97.50% (FA)	95.42% (FA)	96.96%	96.23%	97.62%	95.72%
SHAR	90.66%	94.44% (RP)	90.69% (RP)	93.78%	90.93%	94.84%	90.93%
DNA	94.86%	94.69% (FA)	94.69% (FA)	95.86%	95.36%	93.76%	93.76%

TABLE 4
Neural network parameter comparison

Dataset	MLP	DR (H.A.)	DR (M.C.)	SCANN (H.A.)	SCANN (M.C.)	DR+SCANN (H.A.)	DR+SCANN (M.C.)
SenDrive	56.9k (1×)	2.6k (21.9×)	1140 (49.9×)	10.0k (5.7×)	750 (75.9×)	2.2k (25.9×)	200 (284.5×)
HAR	212.0k (1×)	108.4k (1.9×)	108.4k (1.9×)	5.0k (42.4×)	5.0k (42.4×)	1.0k (212×)	750 (282.7×)
Musk	55.8k (1×)	17.3k (3.2×)	15.5k (3.6×)	22.0k (2.5×)	20.0k (2.8×)	600 (93.0×)	600 (93.0×)
Pendigits	4.9k (1×)	780 (6.3×)	671 (7.3×)	3.2k (1.5×)	3.2k (1.5×)	400 (12.2×)	175 (28.0×)
SatIm	3.8k (1×)	1.1k (3.4×)	1.1k (3.4×)	3.2k (1.5×)	3.2k (1.5×)	1.0k (3.8×)	1.0k (3.8×)
Letter	4.4k (1×)	3.8k (1.1×)	3.8k (1.1×)	3.8k (1.1×)	3.8k (1.1×)	3.7k (1.2×)	3.7k (1.2×)
Seizure	380.9k (1×)	10.5k (36.3×)	616 (618.3×)	3.0k (127.0×)	1.2k (317.4×)	1.8k (211.6×)	75 (5078.7×)
SHAR	214.0k (1×)	127.1k (1.7×)	3.7k (57.8×)	10.0k (21.4×)	800 (267.5×)	1.0k (214.0×)	500 (428.0×)
DNA	24.6k (1×)	22.9k (1.1×)	22.9k (1.1×)	20.0k (1.2×)	200 (123.0×)	300 (82.0×)	300 (82.0×)

TABLE 5
Inference energy consumption comparison (J)

Dataset	MLP	DR (H.A.)	DR (M.C.)	SCANN (H.A.)	SCANN (M.C.)	DR+SCANN (H.A.)	DR+SCANN (M.C.)
SenDrive	4.6e-6	2.1e-7	8.9e-8	8.1e-7	6.1e-8	1.8e-7	1.6e-8
HAR	17.2e-6	8.8e-6	8.8e-6	4.0e-7	4.0e-7	8.1e-8	6.1e-8
Musk	4.5e-6	1.4e-6	1.2e-6	1.8e-6	1.6e-6	4.9e-8	4.9e-8
Pendigits	4.0e-7	6.3e-8	5.4e-8	2.6e-7	2.6e-7	3.2e-8	1.4e-8
SatIm	3.1e-7	8.9e-8	8.9e-8	2.6e-7	2.6e-7	8.1e-8	8.1e-8
Letter	3.6e-7	3.1e-7	3.1e-7	3.1e-7	3.1e-7	3.0e-7	3.0e-7
Seizure	3.1e-5	8.5e-7	5.0e-8	2.4e-7	9.7e-8	1.4e-7	6.1e-9
SHAR	1.7e-5	1.0e-5	3.0e-7	8.1e-7	6.5e-8	8.1e-8	4.0e-8
DNA	2.0e-6	1.8e-6	1.8e-6	1.6e-6	1.6e-8	2.4e-8	2.4e-8

during training. This benefit is not directly available in several other existing automatic architecture synthesis techniques, such as the evolutionary and reinforcement learning based approaches. In those methods, a new architecture, whether generated through mutation and crossover in the evolutionary approach or from the controller in the reinforcement learning approach, needs to be fixed during training and trained from scratch again when the architecture is changed. However, human learning is incremental. Our brain gradually changes based on the presented stimuli. For example, studies of the human neocortex have shown that up to 40 percent of the synapses are rewired every day [59]. Hence, from this perspective, SCANN takes inspiration from how the human brain evolves incrementally. SCANN's dynamic rewiring can be easily achieved through connection growth and pruning.

Comparisons between SCANN and DR+SCANN show that the latter results in a smaller network in nearly all the cases. This is due to the initial step of dimensionality reduction. By mapping data instances into lower dimensions, it reduces the number of neurons in each layer of the neural network, without degrading performance. This helps feed a significantly smaller neural network to SCANN. As a result, DR+SCANN synthesizes smaller networks relative to when only SCANN is used. However, a limitation of SCANN is that it can only evolve feed-forward networks. How to extend SCANN to CNNs and recurrent neural networks is the focus of our future work.

7 Conclusion

In this paper, we proposed a synthesis methodology that can generate compact and accurate neural networks. It solves the problem of having to fix the depth of the network during training that prior synthesis methods suffer from. It is able to evolve an arbitrary feed-forward network architecture with the help of three basic operations: connections growth, neuron growth, and connection pruning. Experiments on the MNIST dataset show that, without loss in accuracy, SCANN generates a 46.3× smaller network than the LeNet-5 Caffe model. Furthermore, by combining dimensionality reduction with SCANN synthesis, we showed significant improvements in the compression power of this framework. Experiments with several other small to medium datasets

show that SCANN and DR+SCANN can provide a good tradeoff between accuracy and energy efficiency in applications where computing resources are limited.

REFERENCES

[1] F. Rosenblatt, *The Perceptron, a Perceiving and Recognizing Automaton Project Para*. Cornell Aeronautical Laboratory, 1957.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105.

[3] G. E. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-R. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[4] S. Jean, K. Cho, R. Memisevic, and Y. Bengio, "On using very large target vocabulary for neural machine translation," *arXiv preprint arXiv:1412.2007*, 2014.

[5] Y. LeCun, Y. Bengio, and G. E. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015.

[6] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2015, pp. 1–9.

[7] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, Inception-ResNet and the impact of residual connections on learning," *arXiv preprint arXiv:1602.07261*, 2016.

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[9] S. Zagoruyko and N. Komodakis, "Wide residual networks," *arXiv preprint arXiv:1605.07146*, 2016.

[10] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2017, pp. 5987–5995.

[11] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten, "Densely connected convolutional networks," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2017, pp. 2261–2269.

[12] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Advances in Neural Information Processing Systems*, 2015, pp. 1135–1143.

[13] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[14] R. Ding, Z. Liu, R. Shi, D. Marculescu, and R. D. Blanton, "LightNN: Filling the gap between conventional deep neural networks and binarized networks," in *Proc. Great Lakes Symposium on VLSI*, 2017, pp. 35–40.

[15] D. Stamoulis, E. Cai, D.-C. Juan, and D. Marculescu, "HyperPower: Power-and memory-constrained hyper-parameter optimization for neural networks," *arXiv preprint arXiv:1712.02446*, 2017.

[16] E. Cai, D.-C. Juan, D. Stamoulis, and D. Marculescu, "NeuralPower: Predict and deploy energy-efficient convolutional neural networks," *arXiv preprint arXiv:1710.05420*, 2017.

[17] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Proc. Advances in Neural Information Processing Systems*, 2016, pp. 4107–4115.

[18] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "ShuffleNet v2: Practical guidelines for efficient CNN architecture design," *arXiv preprint arXiv:1807.11164*, vol. 1, 2018.

[19] X. Dai, P. Zhang, B. Wu, H. Yin, F. Sun, Y. Wang, M. Dukhan, Y. Hu, Y. Wu, Y. Jia, P. Vajda, M. Uyttendaele, and N. K. Jha, "ChamNet: Towards efficient network design through platform-aware model adaptation," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2018.

[20] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[21] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNet v2: Inverted residuals and linear bottlenecks," in *Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition*, 2018, pp. 4510–4520.

[22] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition*, 2018, pp. 8697–8710.

[23] 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.

[24] M. Tan, B. Chen, R. Pang, V. Vasudevan, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," *arXiv preprint arXiv:1807.11626*, 2018.

[25] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proc. ACM Int. Conf. Multimedia*, 2014, pp. 675–678.

[26] J. B. Tenenbaum, V. De Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *Science*, vol. 290, no. 5500, pp. 2319–2323, 2000.

[27] K. Q. Weinberger and L. K. Saul, "An introduction to nonlinear dimensionality reduction by maximum variance unfolding," in *Proc. AAAI*, vol. 6, 2006, pp. 1683–1686.

[28] K. Bhardwaj and R. Marculescu, "Dimensionality reduction via community detection in small sample datasets," in *Proc. Pacific-Asia Conf. Knowledge Discovery and Data Mining*. Springer, 2018, pp. 102–114.

[29] D. K. Agrafiotis, "Stochastic proximity embedding," *J. Computational Chemistry*, vol. 24, no. 10, pp. 1215–1221, 2003.

[30] L. van der Maaten and G. Hinton, "Visualizing data using t-SNE," *J. Machine Learning Research*, vol. 9, pp. 2579–2605, Nov 2008.

[31] L. van der Maaten, E. Postma, and J. Van den Herik, "Dimensionality reduction: A comparative," *J. Machine Learning Research*, vol. 10, pp. 66–71, 2009.

[32] G. F. Miller, P. M. Todd, and S. U. Hegde, "Designing neural networks using genetic algorithms." in *Proc. Int. Conf. Genetic Algorithms*, vol. 89, 1989, pp. 379–384.

[33] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, 2002.

[34] L. Xie and A. Yuille, "Genetic CNN," *arXiv preprint arXiv:1703.01513*, 2017.

[35] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

[36] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," *arXiv preprint arXiv:1707.07012*, 2017.

[37] X. Dai, H. Yin, and N. K. Jha, "NeST: A neural network synthesis tool based on a grow-and-prune paradigm," *arXiv preprint arXiv:1711.02017*, 2017.

[38] ——, "Grow and prune compact, fast, and accurate LSTMs," *arXiv preprint arXiv:1805.11797*, 2018.

[39] T. Zhang, K. Zhang, S. Ye, J. Li, J. Tang, W. Wen, X. Lin, M. Fardad, and Y. Wang, "Adam-ADMM: a unified, systematic framework of structured weight pruning for DNNs," *arXiv preprint arXiv:1807.11091*, 2018.

[40] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," *arXiv preprint arXiv:1611.05128*, 2016.

[41] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Proc. Advances in Neural Information Processing Systems*, 2016, pp. 2074–2082.

[42] T.-J. Yang, A. Howard, B. Chen, X. Zhang, A. Go, M. Sandler, V. Sze, and H. Adam, "NetAdapt: platform-aware neural network adaptation for mobile applications," *Energy*, vol. 41, p. 46, 2018.

[43] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and $<$ 0.5 MB model size," *arXiv preprint arXiv:1602.07360*, 2016.

[44] B. Wu, A. Wan, X. Yue, P. Jin, S. Zhao, N. Golmant, A. Gholaminejad, J. Gonzalez, and K. Keutzer, "Shift: A zero flop, zero parameter alternative to spatial convolutions," *arXiv preprint arXiv:1711.08141*, 2017.

[45] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," in *Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition*, 2018, pp. 6848–6856.

[46] H. Yin, G. Chen, Y. Li, S. Che, W. Zhang, and N. K. Jha, "Hardware-guided symbiotic training for compact, accurate, yet execution-efficient LSTM," *arXiv preprint arXiv:1901.10997*, 2019.

[47] C. Zhu, S. Han, H. Mao, and W. J. Dally, "Trained ternary quantization," *arXiv preprint arXiv:1612.01064*, 2016.

[48] X. Jin, X. Yuan, J. Feng, and S. Yan, "Training skinny deep neural networks with iterative hard thresholding methods," *arXiv preprint arXiv:1607.05423*, 2016.
[49] S. Han, J. Pool, S. Narang, H. Mao, S. Tang, E. Elsen, B. Catanzaro, J. Tran, and W. J. Dally, "DSD: Regularizing deep neural networks with dense-sparse-dense training flow," *arXiv preprint arXiv:1607.04381*, 2016.
[50] D. Sivakumar, "Algorithmic derandomization via complexity theory," in *Proc. ACM Symp. Theory of Computing*, 2002, pp. 619–626.
[51] S. Dasgupta and A. Gupta, "An elementary proof of a theorem of Johnson and Lindenstrauss," *Random Structures & Algorithms*, vol. 22, no. 1, pp. 60–65, 2003.
[52] D. Achlioptas, "Database-friendly random projections," in *Proc. ACM SIGMOD-SIGACT-SIGART Symp. Principles of Database Systems*, 2001, pp. 274–281.
[53] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
[54] X. Dong, S. Chen, and S. Pan, "Learning to prune deep neural networks via layer-wise optimal brain surgeon," in *Proc. Advances in Neural Information Processing Systems*, 2017, pp. 4860–4874.
[55] K. Ullrich, E. Meeds, and M. Welling, "Soft weight-sharing for neural network compression," *arXiv preprint arXiv:1702.04008*, 2017.
[56] D. Dheeru and E. Karra Taniskidou, "UCI Machine Learning Repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml
[57] D. Michie, D. Spiegelhalter, C. Taylor, and J. Campbell, *Machine Learning, Neural and Statistical Classification*. Upper Saddle River, NJ, USA: Ellis Horwood, 1994.
[58] A. O. Akmandor, H. Yin, and N. K. Jha, "Simultaneously ensuring smartness, security, and energy efficiency in Internet-of-Things sensors," in *Proc. IEEE Custom Integrated Circuits Conference*, 2018, pp. 1–8.
[59] J. Hawkins, "What intelligent machines need to learn from the neocortex machines won't become intelligent unless they incorporate certain features of the human brain-here are three of them," *IEEE Spectrum*, vol. 54, no. 6, pp. 34–71, 2017.