NeST: A Neural Network Synthesis Tool Based on a Grow-and-Prune Paradigm

Xiaoliang Dai
Princeton University
xdai@princeton.edu

Hongxu Yin
Princeton University
hongxuy@princeton.edu

Niraj K. Jha
Princeton University
jha@princeton.edu

Abstract

Neural networks (NNs) have begun to have a pervasive impact on various applications of machine learning. However, the problem of finding an optimal NN architecture for large applications has remained open for several decades. Conventional approaches search for the optimal NN architecture through extensive trial-and-error. Such a procedure is quite inefficient. In addition, the generated NN architectures incur substantial redundancy. To address these problems, we propose an NN synthesis tool (NeST) that automatically generates very compact architectures for a given dataset. NeST starts with a seed NN architecture. It iteratively tunes the architecture with gradient-based growth and magnitude-based pruning of neurons and connections. Our experimental results show that NeST yields accurate yet very compact NNs with a wide range of seed architecture selection. For example, for the LeNet-300-100 (LeNet-5) NN architecture derived from the MNIST dataset, we reduce network parameters by $34.1\times$ ($74.3\times$) and floating-point operations (FLOPs) by $35.8\times$ ($43.7\times$). For the AlexNet NN architecture derived from the ImageNet dataset, we reduce network parameters by $15.7\times$ and FLOPs by $4.6\times$. All these results are the current state-of-the-art for these architectures.

1 Introduction

Over the last decade, neural networks (NNs) have begun to revolutionize myriad research domains, such as computer vision, speech recognition, and robotic control [1–6]. Their ability to distill intelligence from a dataset through multi-level abstraction even leads to super-human performance [7]. Thus, NNs are emerging as a new cornerstone of modern artificial intelligence (AI).

The NN architecture derived from a given dataset has a huge impact on its final performance. To illustrate this, in Table 1, we compare several well-known NNs from the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2012-2016 [8]. We describe the architecture in terms of network depth, number of network parameters, and number of connections, and performance in terms of top-5 error rate on the ImageNet dataset. All these contest-winning NNs are obtained using the same back-propagation (BP) algorithm [9] for training weights. Thus, the network architecture becomes their distinguishing characteristic. This architectural diversity leads to substantially different final performance, as can be seen from Table 1.

Though critically important, how to efficiently derive an appropriate NN architecture from a given dataset has remained an open problem, especially for large datasets. Researchers have traditionally derived the NN architecture by sweeping through its architectural parameters and training the corresponding architecture until the point of diminishing returns in its classification performace. This suffers from three major problems:

This work was supported by NSF Grant No. CNS-1617640.

Table 1: Architecture and performance comparison for ILSVRC NNs

Network	Depth	#Parameters	#Connections	Top-5 error
AlexNet [10]	7	61M	0.7B	18.2%
VGG-16 [11]	16	138M	15.5B	7.3%
GoogLeNet [12]	22	7M	1.6B	6.7%
ResNet-152 [13]	152	60M	11.3B	3.6%

1. **Fixed architecture:** Most BP-based methods train weights, not architectures. They only utilize the gradient information in the NN weight space, but keep the NN architecture fixed throughout the entire training process. Thus, such an approach does not lead to a better NN architecture.
2. **Lengthy upgrade:** Searching for an appropriate NN architecture through trial-and-error is extremely inefficient. This problem is exacerbated when NNs get deeper and contain tens of millions of parameters. Each trial on a deep NN can easily consume tens of hours on even the fastest graphical processing units (GPUs). Note that GPUs are currently the main workhorse for NN training. Even with all the available computational power and expended researcher efforts, it still takes years to unveil superior architectures for a given application, such as image classification (e.g., as can be seen from the evolution of AlexNet to VGG, GoogLeNet, and ResNet).
3. **Vast redundancy:** Most NNs are significantly over-parameterized. Even the best-known human-defined NN architectures for image classification (e.g., LeNets [1], AlexNet [10], VGG [11]) suffer from substantial storage and computation redundancy. For example, Han et al. showed that the number of parameters and floating point operations (FLOPs) in AlexNet can be reduced by $9\times$ and $3\times$, respectively, with no loss of accuracy [14].

To address these problems, we propose a novel NN synthesis tool (NeST) that trains both NN weights and architectures. Inspired by the learning mechanism of the human brain, NeST starts NN synthesis from a seed NN architecture (*birth point*). It allows the NN to **grow** connections and neurons based on gradient information (*baby brain*) so that the NN can easily adapt to the problem at hand. Then, it **prunes** away insignificant connections and neurons based on magnitude information (*adult brain*) to avoid redundancy. This enables NeST to generate compact yet accurate NNs. We used NeST to synthesize various compact NNs for the MNIST [1][1] and ImageNet [8][2] datasets. As we show later, NeST leads to drastic reductions in the number of parameters and FLOPs relative to the corresponding state-of-the-art NN baselines while maintaining or slightly improving classification accuracy, hence dramatically cutting memory cost, inference run-time, and energy consumption.

2 Related Work

Various attempts have been made in the past to automate the process of NN architecture selection. These strategies fall into two major categories: (i) evolutionary algorithm (EA), and (ii) structure adaptation (SA). We discuss these approaches next.

2.1 Evolutionary Algorithm

NN architecture selection can be formulated as a search problem in the discrete and complex architecture space, where EAs provide a promising but very unwieldy solution. One of the earliest 'neuro-evolution' methods was proposed by Miller et al. in 1989 [15]. It uses a nature-inspired genetic algorithm to first encode the NN architectures as genes and then improve the architectures through progressive natural selection. Numerous extensions and improvements were made to the 'neuro-evolution' method in the last 30 years. Examples include different encoding methods [16], simultaneous weight and architecture evolution [17], and algorithmic modification and redesign [18–23].

Despite the progress made over the three decades, EAs remain quite inefficient. Their search mechanism involves cumbersome iterations of reproduction, mutation, recombination, and selection. This limits their scalability. For example, EAs cannot tackle the ImageNet dataset with 1.2 million training images. State-of-the-art EAs can only support the CIFAR-10 and CIFAR-100 datesets, each of which contains only 50K 32×32 training images [24].

1. 28×28 handwritten digits, 60K instances for training and 10K for validation.
2. ILSVRC-2012 image classification dataset, 1.2 million instances for training and 50K for validation.

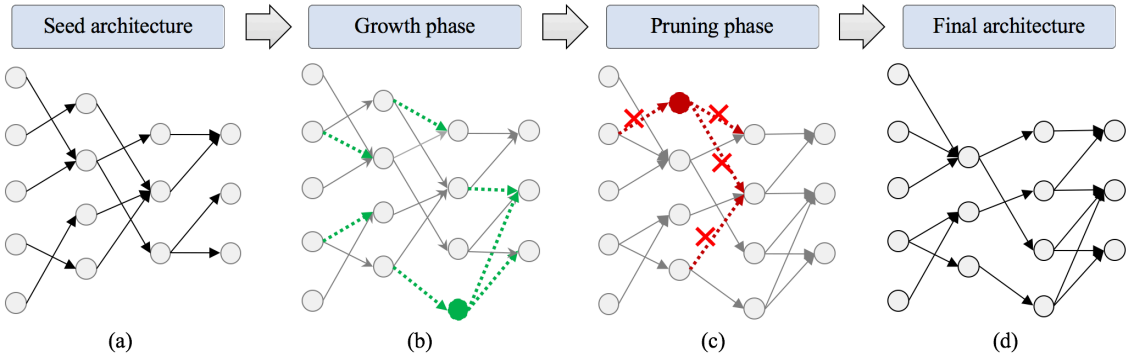

Figure 1: An illustration of the architecture synthesis flow in NeST.

Table 2: Notations and descriptions

Label	Description	Label	Description
L	NN loss function	$\mathbf{W}^l$	weights between $(l-1)^{th}$ and l^{th} layer
x_n^l	output value of n^{th} neuron in l^{th} layer	$\mathbf{b}^l$	biases in l^{th} layer
u_m^l	input value of m^{th} neuron in l^{th} layer	$\mathbf{R}^{d_0 \times d_1 \times d_2}$	d_0 by d_1 by d_2 matrix with real elements

2.2 Structure Adaptation

The SA approach exploits network *clues* (e.g., distribution of weights) to incorporate structural adaptations into the training process. Existing SA methods can be further divided into two major categories [25]:

1. Constructive approach: Starting with a small network, it iteratively **adds** connections/neurons until a desired accuracy is reached [26–34].
2. Destructive approach: Starting with a large network, it iteratively **removes** connections/neurons until a desired size is reached [14, 35–43].

These SA approaches are not yet mature. The constructive approach is inefficient for deep and large-scale NNs. Moreover, it typically yields NNs with significant redundancy. The destructive approach relies heavily on fully-trained accurate NNs as a starting point. However, these starting points can only be obtained through very time-consuming trial-and-error.

3 Synthesis Methodology

In this section, we propose NeST that leverages both constructive and destructive SA approaches through a grow-and-prune paradigm. We first give a high-level overview of NeST, after which we zoom into specific growth and pruning algorithms. Unless otherwise stated, we adopt the notations given in Table 2 to represent various variables.

3.1 Neural Network Synthesis Tool

We first illustrate the NeST approach with Fig. 1. Synthesis begins with an initial seed architecture, typically a sparse and partially connected NN, as shown in Fig. 1(a). Then, it utilizes two sequential phases to synthesize the NN: (i) gradient-based growth phase, and (ii) magnitude-based pruning phase. In the growth phase, the gradient information in the architecture space is used to gradually grow new connections, neurons, and feature maps to achieve the desired accuracy. In the pruning phase, the NN inherits the synthesized architecture and weights from the growth phase and iteratively removes redundant connections and neurons, based on their magnitudes. Finally, NeST comes to rest at a lightweight NN model that incurs no accuracy degradation relative to a fully connected model.

Algorithm 1 shows the details of the grow-and-prune synthesis algorithm. $sizeof$ extracts the total number of parameters and $test$ checks NN accuracy on the validation set. Prior to synthesis, we set constraints on maximum size S and desired accuracy A. We show the major components of the algorithm in Fig. 2. We provide details of these components next.

Algorithm 1 Main architecture synthesis algorithm in NeST

Input: S - maximum size, A - desired accuracy, Net - neural network
$s = sizeof(Net), a = test(Net)$
$train(Net)$
if $s \leq S$ and $a < A$ **then**
  repeat
    Grow connections, neurons, and feature maps
    $Net \leftarrow train(Net)$
    $s = sizeof(Net), a = test(Net)$
  until $s \geq S$ or $a \geq A$
end if
repeat
  Prune connections and neurons
  $Net \leftarrow train(Net)$
  $a = test(Net)$
until $a < A$
Return Net

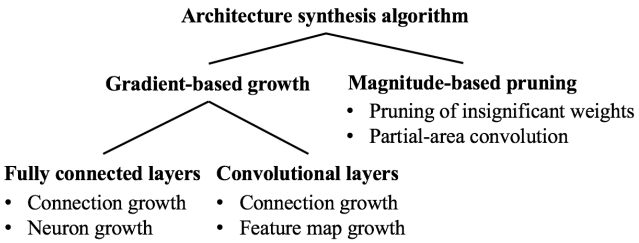

Figure 2: Major components of the NN architecture synthesis algorithm in NeST.

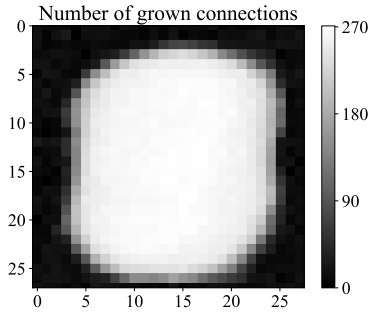

Figure 3: Grown connections from the input layer to the first layer of LeNet-300-100.

3.2 Gradient-based Growth

In this section, we explain how we grow connections, neurons, and feature maps based on gradient information.

3.2.1 Connection Growth

The connection growth algorithm greedily activates useful, but currently 'dormant', connections. We incorporate it in the following learning policy:

Policy 1: Add a connection w iff it can quickly reduce the value of loss function L.

The initial sparse NN seed contains only a small fraction of active connections that can propagate gradient information. To locate the 'dormant' connections that can reduce L effectively, we evaluate $\partial L/\partial w$ for all the 'dormant' connections w (computed either using the whole training set or a large batch). We activate the connections with large gradient magnitudes $|\partial L/\partial w|$.

Policy 1 activates 'dormant' connections iff they are the most efficient at reducing L. To illustrate this policy, we plot the connections grown from the input to the first layer of LeNet-300-100 [1] (for the MNIST dataset) in Fig. 3. The image center has a much higher grown connection density than the image margins. This is consistent with the fact that almost all the MNIST digits are centered.

From a neuroscience perspective, our connection growth algorithm coincides well with the famous Hebbian theory: "Neurons that fire together wire together [44]." We define the stimulation magnitude of the m^{th} presynaptic neuron in the $(l+1)^{th}$ layer and the n^{th} postsynaptic neuron in the l^{th} layer as $\frac{\partial L}{\partial u_m^{l+1}}$ and x_n^l,

Algorithm 2 Neuron growth in the l^{th} layer

Input: α - birth strength, β - growth ratio
Denote: M - number of neurons in the $(l+1)^{th}$ layer, N - number of neurons in the $(l-1)^{th}$ layer, $\mathbf{G} \in R^{M \times N}$ - bridging gradient matrix, avg - operation to extract the mean value of all non-zero elements
Begin
Add a neuron in the l^{th} layer, initialize $\mathbf{w}^{out} = \vec{\mathbf{0}} \in R^M$, $\mathbf{w}^{in} = \vec{\mathbf{0}} \in R^N$
for $1 \leq m \leq M, 1 \leq n \leq N$ **do**
  $G_{m,n} = \frac{\partial L}{\partial u_m^{l+1}} \times x_n^{l-1}$
end for
$thres$ = $(\beta MN)^{th}$ largest element in $abs(\mathbf{G})$
for $1 \leq m \leq M, 1 \leq n \leq N$ **do**
  if $|G_{m,n}| > thres$ **then**
    $\delta w = \sqrt{|G_{m,n}|} \times rand\{1, -1\}$
    $w_m^{out} \leftarrow w_m^{out} + \delta w$, $w_n^{in} \leftarrow w_n^{in} + \delta w \times sgn(G_{m,n})$
  end if
  $\mathbf{w}^{out} \leftarrow \mathbf{w}^{out} \times \alpha \frac{avg(abs(\mathbf{W}^{l+1}))}{avg(abs(\mathbf{w}^{out}))}$, $\mathbf{w}^{in} \leftarrow \mathbf{w}^{in} \times \alpha \frac{avg(abs(\mathbf{W}^{l}))}{avg(abs(\mathbf{w}^{in}))}$
end for
Concatenate network weights $\mathbf{W}$ with $\mathbf{w}^{in}$, $\mathbf{w}^{out}$
End

respectively. Note that the connections activated based on Hebbian theory would have a strong correlation between presynaptic and postsynaptic cells, thus a large value of $|\frac{\partial L}{\partial u_m^{l+1}} x_n^l|$. This is also the magnitude of the gradient of L with respect to w (w is the weight that connects u_m^{l+1} and x_n^l), as shown in Eq. (1).

$$\left|\frac{\partial L}{\partial w}\right| = \left|\frac{\partial L}{\partial u_m^{l+1}} x_n^l\right| \tag{1}$$

Thus, this is mathematically equivalent to Policy 1.

3.2.2 Neuron Growth

Our neuron growth algorithm consists of two major steps: (i) connection establishment and (ii) weight initialization. The neuron growth policy is as follows:

Policy 2: In the l^{th} layer, add a new neuron as a shared intermediate node between existing neuron pairs that have high postsynaptic (x) and presynaptic ($\partial L/\partial u$) neuron correlations (each pair contains one neuron from the $(l-1)^{th}$ layer and the other from the $(l+1)^{th}$ layer). Initialize weights based on batch gradients to reduce the value of L.

Policy 2 also greedily reduces L. Since a newly added neuron connects neurons in the previous and subsequent layers, we only target neuron pairs in these two layers that have strong presynaptic and postsynaptic correlations.

Algorithm 2 incorporates Policy 2 and illustrates the neuron growth iterations in detail. Before adding a neuron to the l^{th} layer, we evaluate the bridging gradient between the neurons at the previous $(l-1)^{th}$ and subsequent $(l+1)^{th}$ layers. We connect the top $\beta \times 100\%$ (β is the growth ratio) correlated neuron pairs through a new neuron in the l^{th} layer. We initialize the weights based on the bridging gradient so that this neuron addition enables gradient descent, thus decreasing the value of L.

We implement a square root rule for weight initialization in order to imitate a BP update on the bridging connection w_b, which connects x_n^{l-1} [output value of the n^{th} neuron in the $(l-1)^{th}$ layer] and u_m^{l+1} [input value of the m^{th} neuron in the $(l+1)^{th}$ layer]. The BP update leads to a change in u_m^{l+1}:

$$|\Delta u_m^{l+1}|_{b.p.} = |x_n^{l-1} \times \delta w_b| = \eta |x_n^{l-1} \times \partial L / w_b| = \eta |x_n^{l-1} \times G_{m,n}| \tag{2}$$

where η is the learning rate. In Algorithm 2, when we connect the newly added neuron (in the l^{th} layer) with x_n^{l-1} and u_m^{l+1}, we initialize their weights to the square root of the magnitude of the bridging gradient, as follows:

$$|\delta w_n^{in}| = |\delta w_m^{out}| = \sqrt{|G_{m,n}|} \tag{3}$$

where δw_n^{in} (δw_m^{out}) is the initialized value of the weight that connects the newly added neuron with x_n^{l-1} (u_m^{l+1}). The weight initialization rule leads to a change in u_m^{l+1} as follows:

$$|\Delta u_m^{l+1}| = |f(x_n^{l-1} \times \delta w_n^{in}) \times \delta w_m^{out}| \tag{4}$$

where f is the neuron activation function. For example, suppose $tanh$ is the activation function. Then,

$$f(x) = tanh(x) \approx x, \text{if } x \ll 1 \tag{5}$$

Since δw_n^{in} and δw_m^{out} are typically very small, the approximation in Eq. (5) leads to Eq. (6).

$$|\Delta u_m^{l+1}| \approx |x_n^{l-1} \times \delta w_n^{in} \times \delta w_m^{out}| = |x_n^{l-1} \times G_{m,n}| = \frac{1}{\eta}|\Delta u_m^{l+1}|_{b.p.} \tag{6}$$

This is linearly proportional to the effect of a BP update. Thus, our weight initialization mathematically imitates a BP update. Though we illustrated the algorithm with the $tanh$ activation function, the weight initialization rule works equally well with other activation functions, such as rectified linear unit (ReLU[1]) and leaky rectified linear unit (Leaky ReLU[2]).

We use a birth strength factor α to strengthen the connections in and out of a newly grown neuron. This mechanism prevents these connections from becoming too weak to survive the pruning phase. Specifically, after square root rule based weight initialization, we scale up the newly added weights by

$$\mathbf{w}^{out} \leftarrow \mathbf{w}^{out} \times \alpha \frac{avg(abs(\mathbf{W}^{l+1}))}{avg(abs(\mathbf{w}^{out}))}, \mathbf{w}^{in} \leftarrow \mathbf{w}^{in} \times \alpha \frac{avg(abs(\mathbf{W}^{l}))}{avg(abs(\mathbf{w}^{in}))} \tag{7}$$

where avg is an operation that extracts the mean value of all non-zero elements. This strengthens new weights. In practice, we find $\alpha > 0.3$ to be an appropriate range.

Our gradient-based weight initialization method easily outperforms the naive approach that just assigns random values to the new weights. Fig. 4 shows the percentage reduction in the value of the loss function L for gradient-based growth versus the naive approach when applied to LeNet-5. Note that as the value of L decreases, it becomes difficult to reduce L with a stochastic approach. Thus, neither method continues to show further improvements.

3.2.3 Growth in the Convolutional Layers

Convolutional layers share the connection growth methodology of Policy 1. However, we use a unique feature map growth algorithm for convolutional layers, which differs from the neuron growth algorithm (Algorithm 2). In a convolutional layer, we convolve input images with kernels to generate feature maps. Thus, to add a feature map, we need to initialize the corresponding set of kernels. We summarize the feature map growth policy as follows:

Policy 3: To add a new feature map to the convolutional layers, randomly generate sets of kernels, and pick the set of kernels that reduces L the most.

Fig. 5 shows the percentage reduction in L for Policy 3 versus the naive approach that initializes the new kernels with random values. Our method again performs far better than the naive approach.

3.3 Magnitude-based Pruning

We prune away insignificant connections and neurons based on the magnitude of weights and outputs, as stated in the following policy:

Policy 4: Remove a connection (neuron) iff the magnitude of the weight (neuron output) is smaller than a pre-defined threshold.

With this policy, a connection/neuron removal only has a minor adverse impact on L, which one can recover from through retraining. Policy 4 has two variants: (i) pruning of insignificant weights, and (ii) partial-area

1. ReLU: $f(x) = max(0, x)$
2. Leaky ReLU: $f(x) = max(0.01x, x)$

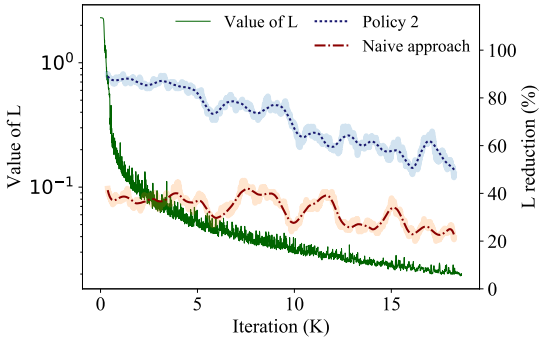

Figure 4: Percentage reduction in L for Policy 2 and naive weight initialization when applied to LeNet-5.

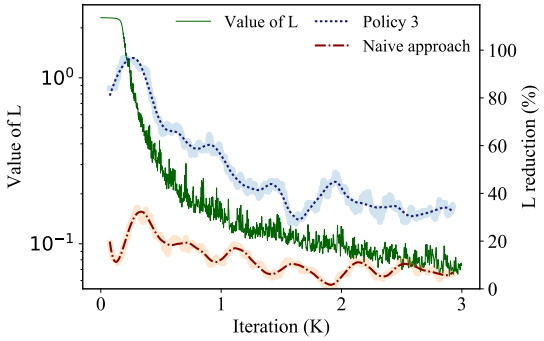

Figure 5: Percentage reduction in L for Policy 3 and naive kernel initialization when applied to LeNet-5.

convolution. Pruning of insignificant weights is targeted at reducing both memory and computation power requirements. Partial-area convolution reduces run-time FLOPs for the NN. We explain these two variants in detail next.

3.3.1 Pruning of Insignificant Weights

Magnitude-based weight pruning was first proposed by Gloger et al. [35] and verified on large-scale NNs by Han et al. [14]. We extend this approach to incorporate the batch normalization technique. Such a technique can reduce the internal covariate shift by normalizing layer inputs. It significantly improves the training speed and behavior and, hence, has been widely applied to large NNs [45]. Consider the l^{th} batch normalization layer [45]:

$$\mathbf{u}^l = [(\mathbf{W}^l\mathbf{x}^{l-1} + \mathbf{b}^l) - \mathbf{E}] \oslash \mathbf{V} = \mathbf{W}^l_*\mathbf{x} + \mathbf{b}^l_* \tag{8}$$

where $\mathbf{E}$ and $\mathbf{V}$ are batch normalization terms, and $\oslash$ depicts the Hadamard (element-wise) division operator. We define effective weights $\mathbf{W}^l_*$ and effective biases $\mathbf{b}^l_*$ as:

$$\mathbf{W}^l_* = \mathbf{W}^l \oslash \mathbf{V}, \mathbf{b}^l_* = (\mathbf{b}^l - \mathbf{E}) \oslash \mathbf{V} \tag{9}$$

We treat connections with small effective weights as insignificant. Pruning of insignificant weights is an iterative process. In each iteration, we only prune the most insignificant weights (e.g., top 1%) for each layer, and then retrain the whole NN to recover its performance.

3.3.2 Partial-area Convolution

In common convolutional neural networks (CNNs), the convolutional layers typically only consume $\sim 5\%$ of the parameters (memory), but contribute to $\sim 90\%$-95% of the total FLOPs (computational load) at inference time [46]. In a convolutional layer, kernels shift and convolve with the entire input image to generate feature maps. This process incurs significant redundancy, since not the whole input image is of interest to a particular kernel. Anwar et al. presented a method to prune all the connections from a not-of-interest input image to a particular kernel [47]. However, such a pruning method is very coarse-grained. It incurs substantial performance degradation.

Instead of discarding an entire image, our proposed partial-area convolution algorithm allows kernels to convolve with only the image areas that are of interest. We refer to such an area as an ***area-of-interest***. We prune away connections to other image areas to avoid incurring unnecessary FLOPs. We illustrate this process in Fig. 6. The green area depicts ***area-of-interest*** in the image, whereas the red area depicts parts that are not of interest. Thus, green connections (solid green lines) are kept, whereas red ones (dashed red lines) are pruned away.

Partial-area convolution pruning is also an iterative process. We present one iteration in Algorithm 3. We first convolve M input images with $M \times N$ convolution kernels, and generate $M \times N$ feature maps, which are stored in the form of a four-dimensional feature map matrix $\mathbf{C}$. We set the pruning threshold $thres$

Algorithm 3 Partial-area convolution

Input: I - M input images, **K** - kernel matrix, **Msk** - feature map mask, γ - pruning ratio
Output: Msk, **F** - N feature maps
Denote: $\mathbf{C} \in R^{M \times N \times P \times Q}$ - Depthwise feature map, $\otimes$ - Hadamard (element-wise) multiplication
for $1 \leq m \leq M, 1 \leq n \leq N$ **do**
  $\mathbf{C}_{m,n}$ = $convolve(\mathbf{I}_m, \mathbf{K}_{m,n})$
end for
$thres$ = $(\gamma MNPQ)^{th}$ largest element in $abs(\mathbf{C})$
for $1 \leq m \leq M, 1 \leq n \leq N, 1 \leq p \leq P, 1 \leq q \leq Q$ **do**
  if $|C_{m,n,p,q}| < thres$ **then**
    $Msk_{m,n,p,q} = 0$
  end if
end for
$\mathbf{C} \leftarrow \mathbf{C} \otimes \mathbf{Msk}$
$\mathbf{F} \leftarrow \Sigma_{m=1}^{M} \mathbf{C}_m$
Return F, **Msk**

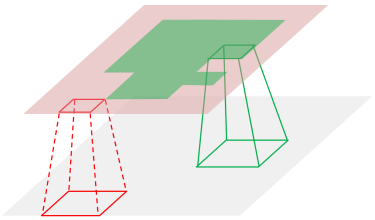

Figure 6: Pruned connections (dashed red lines) and remaining connections (solid green lines) in partial-area convolution.

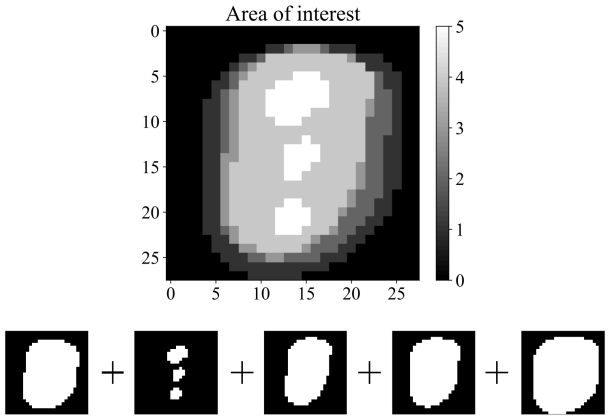

Figure 7: Area-of-interest for five different kernels in the first layer of LeNet-5.

to the $(100\gamma)^{th}$ percentile of all elements in $abs(\mathbf{C})$, where γ is the pruning ratio (we choose $\gamma = 1\%$ in our experiment). We mark the elements whose values are below this threshold as insignificant, and then prune away their input connections. We retrain the whole NN after each pruning iteration. In our current implementation, we utilize a mask **Msk** to disregard the pruned convolution area.

Partial-area convolution enables substantial FLOPs reduction without any performance degradation. For example, we can reduce FLOPs in LeNet-5 by $2.09\times$ when applied to MNIST. Compared to the conventional CNNs that intrinsically force a fixed square-shaped ***area-of-interest*** on all kernels, we allow each kernel to self-explore the preferred shape of its ***area-of-interest***. This exploration cuts down on redundancy and unveils interesting shapes that correspond to different kernels. For example, Fig. 7 shows the ***area-of-interest*** found by the layer-1 kernels in LeNet-5 when applied to MNIST. We observe significant overlaps in the image central area, which most kernels are interested in.

4 Experiemental Results

In this section, we present experimental results obtained by NeST with NN training being performed using Tensorflow [48] on Nvidia GTX 1060 and Tesla P100 GPUs. We use NeST to synthesize compact NNs for the MNIST and ImageNet datasets. We select NN seed architectures based on clues (e.g., depth, kernel size, etc.) from the existing LeNet and AlexNet NN architectures, respectively. NeST exhibits two major advantages:

1. **Wide seed range**: NeST yields high-performance NNs with a wide range of seed architectures. Its greedy nature enables gradient descent in the architecture space. Its ability to start from a wide range

of seed architectures alleviates reliance on human-defined architectures, and offers more freedom to NN designers.

2. **Drastic redundancy removal**: NeST-generated NNs are more compact than the state-of-the-art NNs. Compared to the NN architectures generated with pruning-only methods, NNs generated through our grow-and-prune paradigm have much fewer parameters and require much fewer FLOPs.

Next, we present our experimental results for LeNets on MNIST and AlexNet on ImageNet.

4.1 LeNets on MNIST

We derive the seed architectures from the original LeNet-300-100 and LeNet-5 networks [1]. LeNet-300-100 is a multi-layer perceptron with two hidden layers. LeNet-5 is a CNN with two convolutional layers and three fully connected layers. We use the affine-distorted MNIST dataset [1], on which LeNet-300-100 (LeNet-5) can achieve the lowest error rate of 1.3% (0.8%). We separately discuss our results for the growth and pruning phases and their combination next.

4.1.1 Growth Phase

First, we derive nine (four) seed architectures and term them LeNet-300-100 (LeNet-5) 'birth points'. These seeds contain fewer neurons and connections per layer than the original LeNets. The number of neurons in each layer is the product of a ratio r and the corresponding number in the original LeNets (e.g., the seed architecture for LeNet-300-100 becomes LeNet-270-90 if $r = 0.9$). We randomly initialize only 10% of all the possible connections in the seed architecture. Also, we ensure that all neurons in the network are connected.

We first sweep r for LeNet-300-100 (LeNet-5) from 0.2 (0.5) to 1.0 (1.0) with a step-size of 0.1 (0.17), and then grow the NN architectures from these seeds. We study the impact of these seeds on the GPU time for growth and post-growth NN sizes under the same target accuracy (this accuracy is typically the state-of-the-art accuracy obtained previously for that architecture). We summarize the results for LeNet-300-100 and LeNet-5 in Fig. 8 and Fig. 9, respectively. All the models in each figure share the same target accuracy. We can make two major observations for the growth phase as follows:

- **Observation 1:** There is a trade-off between growth time and post-growth NN size. Smaller seed architectures often lead to smaller post-growth NN sizes, but at the expense of a higher growth time. We will later show that smaller seeds and thus smaller post-growth NN sizes are better, since they also lead to smaller final NN sizes.
- **Observation 2:** When the post-growth NN size saturates due to the full exploitation of the synthesis freedom for a target accuracy, a smaller seed is no longer beneficial. Hence, there is a minimum post-growth NN size associated with a given target accuracy constraint, as evident from the flat left end of the dashed curves in Fig. 8 and Fig. 9. Beyond this point, using a smaller seed architecture only increases growth time, but does not further reduce the post-growth NN size.

4.1.2 Pruning Phase

Next, we prune the post-growth LeNet NNs to remove their redundant neurons/connections. We show the post-pruning NN sizes and compression ratios for LeNet-300-100 (LeNet-5) for the different seeds in Fig. 10 (Fig. 11), where the compression ratio is the quotient of pre-pruning NN size divided by the post-pruning NN size. Again, all the models in each figure have the same target accuracy. We have two major observations for the pruning phase as follows:

- **Observation 1:** Larger the pre-pruning NN, larger is its compression ratio. This is because larger pre-pruning NNs have a larger number of weights and thus also higher redundancy.
- **Observation 2:** Larger the pre-pruning NN, larger is its post-pruning NN. In most cases, NNs with a larger number of weights are still larger after pruning. This is a fundamental limitation of the pruning-only approach. Thus, to synthesize a more compact NN, one should choose a smaller seed architecture (growth phase **Observation 1**) within an appropriate range (growth phase **Observation 2**).

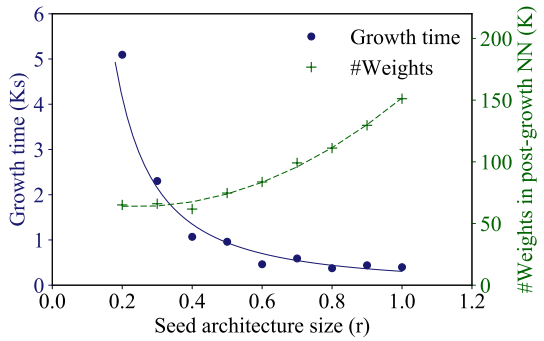

Figure 8: Growth time vs. post-growth NN size trade-off for various seed architectures for LeNet-300-100 to reach the target 1.3% error rate.

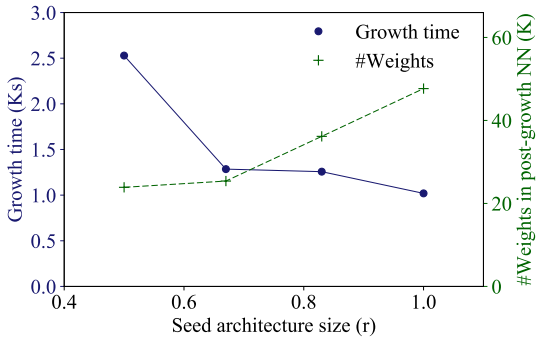

Figure 9: Growth time vs. post-growth NN size trade-off for various seed architectures for LeNet-5 to reach the target 0.8% error rate.

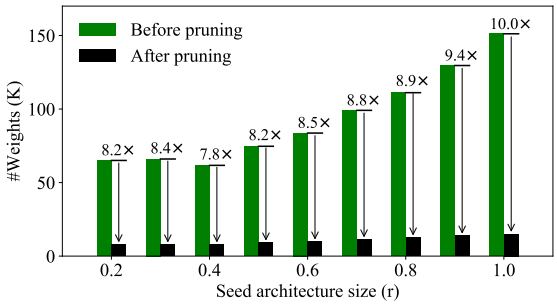

Figure 10: Compression ratio and final NN size for different LeNet-300-100 seed architectures.

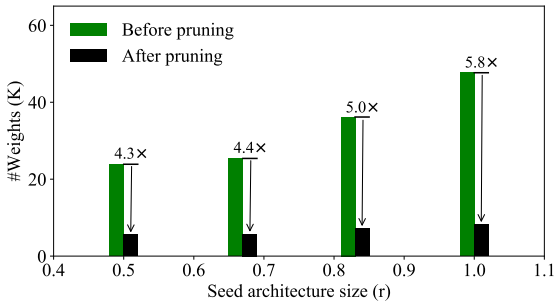

Figure 11: Compression ratio and final NN size for different LeNet-5 seed architectures.

4.1.3 Results and discussions

Table 3(a) and Table 3(b) show the smallest NN models we could synthesize for LeNet-300-100 and LeNet-5, respectively. In these tables, Conv% refers to the percentage of ***area-of-interest*** over a full image for partial-area convolution, and Act% refers to the percentage of non-zero activations (the average percentage of neurons with non-zero output values per inference).

Our results are better than all the previous state-of-the-art. We compare our results against related results from the literature in Table 4. Without any loss of accuracy, we were able to reduce the number of connections and FLOPs of LeNet-300-100 (LeNet-5) by $34.1\times$ ($74.3\times$) and $35.8\times$ ($43.7\times$), respectively, relative to the baseline Caffe model [49]. Our results also substantially outperform other reference models from various

Table 3: Smallest synthesized LeNets

(a) LeNet-300-100 (error rate 1.29%)

Layer	#Weights	Act%	FLOPs
fc1	7032	46%	14.1K
fc2	718	71%	0.7K
fc3	94	100%	0.1K
Total	7844	N/A	14.9K

(b) LeNet-5 (error rate 0.77%)

Layer	#Weights	Conv%	Act%	FLOPs
conv1	74	39%	89%	45.2K
conv2	749	41%	57%	54.4K
fc1	4151	N/A	79%	4.7K
fc2	632	N/A	58%	1.0K
fc3	166	N/A	100%	0.2K
Total	5772	N/A	N/A	105K

Table 4: Different inference models for MNIST

Model	Error	#Parameters	FLOPs
Linear classifier [1]	8.40%	4K	8K
RBF network [1]	3.60%	794K	1588K
Polynomial classifier [1]	3.30%	40K	78K
K-nearest neighbors [1]	3.09%	47M	94M
SVMs (reduced set) [51]	1.10%	650K	1300K
Caffe model (LeNet-300-100) [49]	1.60%	266K	532K
Layer-wise pruning (LeNet-300-100) [52]	1.96%	4K	8K
Network pruning (LeNet-300-100) [14]	1.59%	22K	43K
Our LeNet-300-100	**1.58%**	**3.8K**	**6.7K**
Our LeNet-300-100	**1.29%**	**7.8K**	**14.9K**
Caffe model (LeNet-5) [49]	0.8%	431K	4586K
Layer-wise pruning (LeNet-5) [52]	1.66%	4K	199K^a
Network pruning (LeNet-5) [14]	0.77%	35K	734K
Our LeNet-5	**0.77%**	**5.8K**	**105K**

a. estimated value

design perspectives. Note that LeNet-5 Caffe model is a variant of the original LeNet-5 proposed in [1]. It contains one less layer, $7.3\times$ more parameters, and $6.3\times$ more connections.

In our implementation, we incorporate an activation function shift mechanism to improve accuracy and reduce FLOPs. In the growth phase, we use Leaky ReLU as the activation function to improve accuracy. This is because Leaky ReLU alleviates the 'dying ReLU' problem (i.e., when an inactive ReLU neuron is stuck at a perpetually inactive state since no gradients flow backward through it) and leads to better performance [50]. Then, we keep the network weights, change all the activation functions to ReLU, and retrain the NN. Finally, we prune the network with the ReLU activation function, where we take advantage of ReLU's zero outputs to reduce FLOPs.

4.2 AlexNet on ImageNet

Next, we use NeST to synthesize an NN for the ILSVRC 2012 image classification dataset [8]. We initialize a much simpler seed architecture base on the AlexNet proposed in [46]. The original AlexNet contains 64, 192, 384, 384, and 256 feature maps in the five convolutional layers, and 4096, 4096, and 1000 neurons in its three fully connected layers. Our seed architecture contains only 60, 140, 240, 210, and 160 feature maps in the five convolutional layers, and 3200, 1600, and 1000 neurons in the fully connected layers. We randomly activate 30% of all the possible connections in the seed architecture. We ensure that all neurons are connected. We use batch normalization instead of dropout in our implementation, since batch normalization can act as a regularizer and eliminate the need for dropout [45].

Table 5 illustrates the evolution of an AlexNet seed in the synthesis flow. The seed only contains 8.4M parameters. This number increases to 28.3M after the growth phase, and then decreases to 3.9M after the pruning phase. Finally, NeST synthesizes an AlexNet-based NN model that only contains 3.9M parameters and 325M FLOPs at a top-1 error rate of 42.76%.

Table 6 compares the model synthesized by NeST with various AlexNet-based inference models. We choose the AlexNet Caffe model (42.78% top-1 error rate, 19.73% top-5 error rate) as our baseline, the same as the baseline chosen by Han et al. [14]. Our grow-and-prune synthesis paradigm again outperforms all the pruning-only methods listed in Table 6. This is due to the fundamental limitation of pruning methods: a larger pre-pruning NN yields a larger post-pruning NN. Thus, all pruning methods inherit a certain amount of suboptimality associated with the original large NNs.

5 Discussions

In this section, we discuss the inspirations from the human brain behind our synthesis methodology.

The human brain has continuously provided serendipitous inspirations for modern AI. Examples include the fundamental idea of a neuron, layered NN structure, and even convolution kernels. Our synthesis methodology incorporates three inspirations from the human brain.

Table 5: Synthesized AlexNet (error rate 42.76%)

Layers	#Parameters	#Parameters	#Parameters	Conv%	Act%	FLOPs
	Seed	Post-Growth	Post-Pruning			
conv1	7K	21K	17K	92%	87%	97M
conv2	65K	209K	107K	91%	82%	124M
conv3	95K	302K	164K	88%	49%	40M
conv4	141K	495K	253K	86%	48%	36M
conv5	105K	355K	180K	87%	56%	25M
fc1	5.7M	19.9M	1.8M	N/A	49%	2.0M
fc2	1.7M	5.3M	0.8M	N/A	47%	0.8M
fc3	0.6M	1.7M	0.5M	N/A	100%	0.5M
Total	8.4M	28.3M	**3.9M**	N/A	N/A	**325M**

Table 6: Different AlexNet-based inference models for ImageNet

Model	ΔTop-1 error	ΔTop-5 error	#Parameters (M)	FLOPs (B)
Baseline AlexNet [10]	0.0%	0.0%	61 (1.0×)	1.5 (1.0×)
Data-free pruning [53]	+1.62%	-	39.6 (1.5×)	1.0 (1.5×)
Memory-bounded [54]	+1.62%	-	15.2 (4.0×)	-
SVD [55]	+1.24%	+0.83%	11.9 (5.1×)	-
Layer-wise pruning [52]	+0.33%	+0.28%	6.7 (9.1×)	0.5 (3.0×)
Fastfood-16-AD [56]	+0.12%	-	16.4 (3.7×)	1.4 (1.1×)
Network pruning [14]	-0.01%	-0.06%	6.7 (9.1×)	0.5 (3.0×)
Our AlexNet	**-0.02%**	**-0.06%**	**3.9 (15.7×)**	**0.33 (4.6×)**

First, the number of synaptic connections in the brain varies at different human ages [57, 58]. It rapidly increases upon the baby's birth, peaks after a few months, and decreases steadily thereafter. An NN experiences a very similar learning process in NeST. The initial seed NN is simple and sparse, akin to a baby's brain at **birth point**. In the growth phase, it rapidly grows its connections and neurons based on outside information, thus reacting to this information in a manner similar to how a **baby brain** reacts. In the pruning phase, it reduces the number of synaptic connections to rid itself of the vast redundancy, which is akin to how a baby brain matures into an **adult brain**. To have a clearer view of this trend, we show the plot of the number of connections in LeNet-300-100 along its synthesis lifespan in Fig. 12. It can be seen that this curve shares a very similar pattern to the number of synapses in the human brain as it evolves [59].

Second, most learning processes in our brain result from rewiring of synapses between neurons. Our brain grows and prunes away a large amount (up to 40%) of synaptic connections every day [60]. NeST wakes up new connections, thus effectively rewiring more neurons pairs in the learning process. Thus, it mimics the 'learning through rewiring' mechanism of human brains.

Third, only a small fraction of neurons are active at any given time in human brains. This is known as the sparse neuron response phenomena [60]. This mechanism enables the human brain to operate at an ultra-low power (20 Watts). However, fully connected NNs contain a substantial amount of insignificant

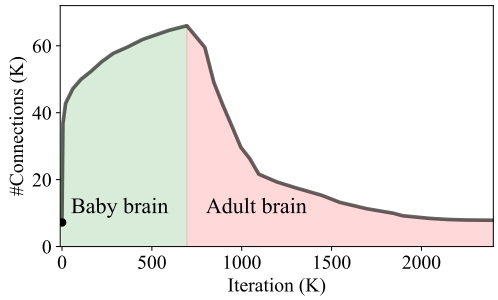

Figure 12: Number of connections vs. synthesis iteration for LeNet-300-100.

neuron responses per inference. To address this problem, we include a magnitude-based neuron/connection pruning algorithm in NeST to remove the redundancy, thus achieving sparsity and compactness. This leads to huge storage and computation reductions.

6 Conclusions

In this paper, we proposed a synthesis tool, NeST, to synthesize compact yet accurate NNs. NeST starts from a sparse seed architecture, adaptively adjusts the architecture through gradient-based growth and magnitude-based pruning, and finally arrives at a compact NN with high accuracy. For LeNet-300-100 (LeNet-5) that targets the MNIST dataset, we reduced the number of network parameters by $34.1\times$ ($74.3\times$) and FLOPs by $35.8\times$ ($43.7\times$). For AlexNet, we reduced the number of network parameters by $15.7\times$ and FLOPs by $4.6\times$. All our synthesis results for the LeNets and AlexNet constitute the current state-of-the-art.

References

[1] 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.

[2] A. Y. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, and E. Liang, "Autonomous inverted helicopter flight via reinforcement learning," in *Experimental Robotics IX*. Springer, 2006, pp. 363–372.

[3] A. Graves, A.-R. Mohamed, and G. E. Hinton, "Speech recognition with deep recurrent neural networks," in *Proc. IEEE Int. Conf. Acoustics, Speech and Signal Processing*, 2013, pp. 6645–6649.

[4] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proc. Advances in Neural Information Processing Systems*, 2014, pp. 3104–3112.

[5] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[6] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *Proc. Advances in Neural Information Processing Systems*, vol. 2011, no. 2, 2011, p. 5.

[7] Y. LeCun, Y. Bengio, and G. E. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[8] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and F.-F. Li, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[9] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Cognitive Modeling*, vol. 5, no. 3, p. 1, 1988.

[10] 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.

[11] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[12] 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.

[13] 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.

[14] 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.

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

[16] M. Mandischer, "Representation and evolution of neural networks," in *Artificial Neural Nets and Genetic Algorithms*. Springer, 1993, pp. 643–649.

[17] J. R. Koza and J. P. Rice, "Genetic generation of both the weights and architecture for a neural network," in *Proc. IEEE Int. Conf. Neural Networks*, vol. 2, 1991, pp. 397–404.

[18] N. Dodd, "Optimisation of network structure using genetic techniques," in *Proc. Int. Joint Conf. on Neural Networks*, 1990, pp. 965–970.

[19] S. A. Harp, T. Samad, and A. Guha, "Designing application-specific neural networks using the genetic algorithm." in *Proc. Advances in Neural Information Processing Systems*, vol. 2, 1989, pp. 447–454.

[20] H. Kitano, "Designing neural networks using genetic algorithms with graph generation system," *Complex Systems*, vol. 4, no. 4, pp. 461–476, 1990.

[21] W. Schiffmann, M. Joost, and R. Werner, "Performance evaluation of evolutionarily created neural network topologies," in *Proc. Int. Conf. Parallel Problem Solving from Nature*, 1990, pp. 274–283.

[22] J. Santos and R. J. Duro, "Evolutionary generation and training of recurrent artificial neural networks," in *Proc. IEEE Conf. Evolutionary Computation*, 1994, pp. 759–763.

[23] K. M. Salama and A. M. Abdelbar, "Learning neural network structures with ant colony algorithms," *Swarm Intelligence*, vol. 9, no. 4, pp. 229–265, 2015.

[24] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, Q. Le, and A. Kurakin, "Large-scale evolution of image classifiers," *arXiv preprint arXiv:1703.01041*, 2017.

[25] D. Elizondo and E. Fiesler, "A survey of partially connected neural networks," *Int. J. of Neural Systems*, vol. 8, pp. 535–558, 1997.

[26] M. Mezard and J.-P. Nadal, "Learning in feedforward layered networks: The tiling algorithm," *J. of Physics A: Mathematical and General*, vol. 22, no. 12, p. 2191, 1989.

[27] T. Ash, "Dynamic node creation in backpropagation networks," *Connection Science*, vol. 1, no. 4, pp. 365–375, 1989.

[28] S. E. Fahlman and C. Lebiere, "The cascade-correlation learning architecture," in *Proc. Advances in Neural Information Processing Systems*, 1990, pp. 524–532.

[29] O. Aran, O. T. Yildiz, and E. Alpaydin, "An incremental framework based on cross-validation for estimating the architecture of a multilayer perceptron," *Int. J. Pattern Recognition and Artificial Intelligence*, vol. 23, no. 02, pp. 159–190, 2009.

[30] J. H. Friedman and W. Stuetzle, "Projection pursuit regression," *J. American Statistical Association*, vol. 76, no. 376, pp. 817–823, 1981.

[31] W. Gloger and G. Häusler, "Neural nets with reduced connectivity for the processing of large pictures," *Int. J. Optical Computing*, vol. 2, p. 425, 1993.

[32] V. Honavar and L. Uhr, *Experimental results indicate that generation, local receptive fields and global convergence improve perceptual learning in connectionist networks*. Technical Rep., University of Wisconsin-Madison, Computer Sciences Department, 1988.

[33] M. Hagiwara, "Novel backpropagation algorithm for reduction of hidden units and acceleration of convergence using artificial selection," in *Proc. Int. Joint Conf. Neural Networks*, 1990, pp. 625–630.

[34] Y. Hirose, K. Yamashita, and S. Hijiya, "Back-propagation algorithm which varies the number of hidden units," *Neural Networks*, vol. 4, no. 1, pp. 61–66, 1991.

[35] T. M. Nabhan and A. Y. Zomaya, "Toward generating neural network structures for function approximation," *Neural Networks*, vol. 7, no. 1, pp. 89–99, 1994.

[36] Y. Chauvin, "A back-propagation algorithm with optimal use of hidden units." in *Proc. Advances in Neural Information Processing Systems*, vol. 1, 1988, pp. 519–526.

[37] S. J. Hanson and L. Y. Pratt, "Comparing biases for minimal network construction with back-propagation," in *Proc. Advances in Neural Information Processing Systems*, 1989, pp. 177–185.

[38] C. Ji, R. R. Snapp, and D. Psaltis, "Generalizing smoothness constraints from discrete samples," *Neural Computation*, vol. 2, no. 2, pp. 188–197, 1990.

[39] B. Hassibi, D. G. Stork *et al.*, "Second order derivatives for network pruning: Optimal brain surgeon," in *Proc. Advances in Neural Information Processing Systems*, 1993, pp. 164–164.

[40] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Proc. Advances in Neural Information Processing Systems 2*, 1990, pp. 598–605.

[41] H. H. Thodberg, "Improving generalization of neural networks through pruning," *Int. J. Neural Systems*, vol. 1, no. 04, pp. 317–326, 1991.

[42] M. Frean, "The upstart algorithm: A method for constructing and training feedforward neural networks," *Neural Computation*, vol. 2, no. 2, pp. 198–209, 1990.

[43] A. S. Weigend, D. E. Rumelhart, and B. A. Huberman, "Generalization by weight-elimination with application to forecasting." in *Proc. Advances in Neural Information Processing Systems*, vol. 90, 1990, pp. 875–882.

[44] S. Lowel and W. Singer, "Selection of intrinsic horizontal connections in the visual cortex by correlated neuronal activity," *Science*, vol. 255, no. 5041, p. 209, 1992.

[45] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conf. Machine Learning*, 2015, pp. 448–456.

[46] A. Krizhevsky, "One weird trick for parallelizing convolutional neural networks," *arXiv preprint arXiv:1404.5997*, 2014.

[47] S. Anwar and W. Sung, "Compact deep convolutional neural networks with coarse pruning," *arXiv preprint arXiv:1610.09639*, 2016.

[48] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[49] 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.

[50] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," *arXiv preprint arXiv:1505.00853*, 2015.

[51] C. J. Burges and B. Schölkopf, "Improving the accuracy and speed of support vector machines," in *Proc. Advances in Neural Information Processing Systems*, 1997, pp. 375–381.

[52] X. Dong, S. Chen, and S. J. Pan, "Learning to prune deep neural networks via layer-wise optimal brain surgeon," *arXiv preprint arXiv:1705.07565*, 2017.

[53] S. Srinivas and R. V. Babu, "Data-free parameter pruning for deep neural networks," *arXiv preprint arXiv:1507.06149*, 2015.

[54] M. D. Collins and P. Kohli, "Memory bounded deep convolutional networks," *arXiv preprint arXiv:1412.1442*, 2014.

[55] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *Proc. Advances in Neural Information Processing Systems*, 2014, pp. 1269–1277.

[56] Z. Yang, M. Moczulski, M. Denil, N. de Freitas, A. Smola, L. Song, and Z. Wang, "Deep fried ConvNets," in *Proc. IEEE Int. Conf. Computer Vision*, 2015, pp. 1476–1483.

[57] B. Pakkenberg and H. J. G. Gundersen, "Neocortical neuron number in humans: Effect of sex and age," *J. Comparative Neurology*, vol. 384, no. 2, pp. 312–320, 1997.

[58] J. J. Park, Y. Tang, I. Lopez, and A. Ishiyama, "Age-related change in the number of neurons in the human vestibular ganglion," *J. Comparative Neurology*, vol. 431, no. 4, pp. 437–443, 2001.

[59] G. Leisman, R. Mualem, and S. K. Mughrabi, "The neurological development of the child with the educational enrichment in mind," *Psicología Educativa*, vol. 21, no. 2, pp. 79–96, 2015.

[60] J. Hawkins, "Machines won't become intelligent unless they incorporate certain features of the human brain," *IEEE Spectrum*, vol. 54, no. 6, pp. 34–71, Jun. 2017.