Neural Network Simplification using a Progressive Barrier Based Approach

by

Tien-Ju Yang

B.S., National Taiwan University (2010)
M.S., National Taiwan University (2012)

Submitted to
the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of
Master of Science in Electrical Engineering and Computer Science

at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2018

© 2018 Tien-Ju Yang. All rights reserved.

The author hereby grants to MIT permission to reproduce and to distribute publicly paper and electronic copies of this thesis document in whole or in part in any medium now known or hereafter created.

Signature redacted

May 23, 2018

Vivienne Sze
Associate Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Signature redacted

Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students
Neural Network Simplification using a Progressive Barrier Based Approach

by

Tien-Ju Yang

Submitted to the Department of Electrical Engineering and Computer Science
on May 23, 2018
in partial fulfillment of the requirements for the degree of
Master of Science in Electrical Engineering and Computer Science

Abstract

Neural networks are indispensable to state-of-the-art artificial intelligence algorithms. However, its high accuracy comes at the cost of high computational complexity. This leads to the high operating cost of data centers and also hinders its deployment on mobile devices. In this thesis, we propose an algorithm to address this problem. The proposed algorithm uses progressive barriers to automatically and progressively simplify a pre-trained neural network until the target complexity is met while maximizing the accuracy. Along with the neural network that meets the target complexity, the algorithm also generates a family of simplified networks with different accuracy-complexity trade-offs, which allows for dynamic network selection and further study. Experiment results show that the algorithm achieves better accuracy-complexity trade-offs on a highly compact MobileNet architecture, compared with state-of-the-art automated network simplification approaches. For image classification on the ImageNet dataset, the algorithm reduces the number of multiply-accumulate operations by $1.68\times$ while achieving $0.9\%$ higher accuracy.

Thesis Supervisor: Vivienne Sze
Title: Associate Professor of Electrical Engineering and Computer Science
Acknowledgments

The thesis would not have been possible without the help and support from many people. First of all, I would like to express my deepest gratitude to my advisor, Professor Vivienne Sze, for giving me the opportunity to join her group and her guidance throughout my research journey at MIT. I also want to thank my collaborators from Google for the thought-provoking comments and discussions. To my amazing lab-mates, I should say that I am incredibly lucky to work with them and thank them for making the lab an enjoyable place. Finally, I would like to thank my family for the support and the encouragement they have provided during ups and downs of my study and research.
# Contents

1 Introduction ................................................. 15
   1.1 Introduction ........................................ 15
   1.2 Artificial Neural Networks ............................. 16
   1.3 Our Contributions ................................... 22

2 Overview of Neural Network Simplification Algorithms .... 25
   2.1 Network Pruning ....................................... 25
   2.2 Quantization .......................................... 27
   2.3 Compact Network Architecture .......................... 29
   2.4 Knowledge Distillation ................................ 32

3 The Proposed Progressive Barrier Based Algorithm for Simplifying Neural Networks .... 33
   3.1 Problem Formulation ................................... 34
   3.2 Algorithm Overview ................................... 35
   3.3 Algorithm Details ..................................... 36
      3.3.1 Choose Number of Filters ......................... 37
      3.3.2 Choose Which Filters ............................... 37
      3.3.3 Short-/Long-Term Fine-Tune ....................... 37

4 Experiment Results ........................................... 39
   4.1 Detailed Settings ...................................... 40
      4.1.1 Algorithm Configuration ........................... 40
4.1.2 Training Configuration ........................................ 40
4.2 Comparison with Benchmark Algorithms ....................... 41
  4.2.1 Simplifying 50% MobileNet (128) .......................... 41
  4.2.2 Simplifying 100% MobileNet (224) ......................... 42
4.3 Ablation Studies .................................................. 43
  4.3.1 Layer Sensitivity to Simplification ......................... 43
  4.3.2 Impact of Short-Term Fine-Tuning ......................... 44
  4.3.3 Impact of Long-Term Fine-Tuning ......................... 44
4.4 Analysis of Simplified Network Architectures ................. 45

5 Conclusion ......................................................... 47

Bibliography .......................................................... 49
List of Figures

1-1 The traditional machine learning algorithms can achieve higher accuracy than neural networks when the data size is small. However, the accuracy does not scale well with the amount of available data. This drawback limits the achievable accuracy of traditional machine learning methods. In contrast, neural networks can take advantage of a significant amount of data and outperform traditional methods when the data set is large enough. (The figure was adapted from [1].)  

1-2 The input stimuli are processed by a series of visual areas. As the stimuli pass through these areas, the size of the receptive field is increased and more complex features are extract [2].  

1-3 This figure illustrates a fully-connected neural network with only one hidden layer. (The figure was adapted from [3].)  

1-4 This figure illustrates a 2-D convolution, which is widely used in image filtering [4].  

1-5 This figure illustrates a high-dimensional convolution, which is used in convolutional neural network [4].  

1-6 This figure shows the typical architecture of modern convolutional neural networks on the image classification application [4].
2-1 This figure illustrates three different network pruning strategies, where each 3\times3 block is a filter in a layer. Weight pruning prunes individual weights from the filters, and filter pruning removes entire filters. Structured pruning prunes weights with a particular structure, and filter pruning can be viewed as a special case of structured pruning.

2-2 This figure illustrates various methods of quantization [5, 6].

2-3 A large filter can be decomposed into multiple smaller filters.

2-4 Knowledge distillation matches the class scores of a small neural network (student) to that of a large neural network or a network ensemble (teacher).

3-1 This figure visualizes the algorithm flow of NetReduce. At each iteration, NetReduce reduces the complexity by simplifying (i.e., removing filters from) one layer. In order to maximize accuracy, it tries to simplify each layer individually and picks the simplified network that has the highest accuracy. Once the target complexity is met, the chosen network is then fine-tuned again until convergence.

4-1 The figure compares NetReduce (simplifying 50\% MobileNet (128)) with the MobileNet family [7] and MorphNet [8].

4-2 The figure compares NetReduce (simplifying 100\% MobileNet (224)) with the MobileNet family [7] and ADC [9].

4-3 This figure shows the sensitivity of simplifying each layer of 50\% MobileNet (128) to give an around 6.8\% drop in the number of MACs. The simplified networks are fine-tuned for 150 epochs.

4-4 This figure shows the accuracy with and without performing short-term fine-tuning (SFT) when simplifying 50\% MobileNet (128) (without performing long-term fine-tuning).
4-5 The comparison between with and without performing long-term fine-tuning (LFT) when simplifying 50% MobileNet (128). Although the short-term fine-tuning preserves the accuracy well, the long-term fine-tuning gives the extra 3% in average (from 1.9% to 4.5%).

4-6 The comparison between 25% MobileNet (128) and the simplified network starting from 50% MobileNet (128). Both of them have a similar number of MACs.

4-7 The comparison between 75% MobileNet (224) and the simplified network starting from 100% MobileNet (224). Both of them have a similar number of MACs.
List of Tables

1.1 The table compares the computational complexity of a traditional algorithm with the CNN alternatives on the image classification application [10]. .................................................. 22

4.1 A pairwise comparison between NetReduce (simplifying 50% MobileNet (128)) and the other two benchmark algorithms, the MobileNet family [7] and MorphNet [8]. We compare the number of MACs at similar accuracy. .................................................. 41

4.2 A pairwise comparison between NetReduce (simplifying 100% MobileNet (224)) and the other two benchmark algorithms, the MobileNet family [7] and ADC [9]. We compare the number of MACs at similar accuracy. .................................................. 42
Chapter 1

Introduction

1.1 Introduction

How to create intelligent machines is a long-lasting question. One natural way to address this question is to mimic humans' behaviors because humans are the most intelligent 'machines' we know so far. After decades of exploration, machine learning has been developed and proven to be an effective methodology. Machine learning possesses an important ability that humans have — learning from data. We are not born with all the knowledge programmed in our brain. As we grow up, we keep learning from the data sensed and collected from the surroundings. Through this process, humans gradually become capable of performing different tasks. Similarly, machine learning is able to learn how to perform a task from the task-specific data. The learning process is called *training*. With training, laboriously creating a distinct, custom program to handle each individual task is not needed. Moreover, the algorithm and the task are decoupled, and the same algorithm can be used on different domains if the data is available.

However, the traditional machine learning algorithms suffer from the following problems:

- The algorithms do not learn well directly from the raw data. They require human experts to manually craft features to present the information in an easier-
Figure 1-1: The traditional machine learning algorithms can achieve higher accuracy than neural networks when the data size is small. However, the accuracy does not scale well with the amount of available data. This drawback limits the achievable accuracy of traditional machine learning methods. In contrast, neural networks can take advantage of a significant amount of data and outperform traditional methods when the data set is large enough. (The figure was adapted from [1].)

- The algorithms do not scale well with the amount of data available (Fig. 1-1). The accuracy saturates very fast with a small amount of training data. With the Internet, collecting data becomes much easier, but the algorithms cannot take advantage of it. This limits the achievable accuracy.

1.2 Artificial Neural Networks

To solve the problems mentioned above, researchers took one step ahead to learn from the architecture of human brains and created artificial neural networks.\(^1\) Although the details of how the brain works are not fully understood, it is generally believed that the basic unit of brains is the neuron. There are approximately 86 billion neurons in the average human brain. The neurons are connected together and cooperate with each other to perform a task. Each neuron takes in the outputs of some of the other neurons.

\(^1\)This section contains excerpts of our co-authored paper [4].
neurons, performs a computation, and generates an output for other neurons. The output of a neural is called activation. One important operation of the computation is scaling the input activations and summing them up. Each input has the corresponding scaling factor, and the scaling factor is called weight. The brain is believed to learn via modifying the weights to achieve different responses to different input signals. Moreover, instead of directly outputting the weighted sum, a neuron only generates an activation when the weighted sum is above a threshold. This behavior is usually modeled as a non-linear activation function. Therefore, a neuron can be modeled by

\[ y_j = f(\sum_{i=0}^{n} w_{ij} \times x_i + b), \]  

where \( x_i \) and \( y_j \) are the input and output activation respectively, \( w_{ij} \) is a weight and \( b \) is a bias. \( f(\cdot) \) is a non-linear activation function and a popular choice is the rectified linear unit (ReLU) [11] due to its effectiveness and simplicity. ReLU bypasses the input to the output when it is larger than zero. Otherwise, ReLU outputs zero. Therefore, the output of ReLU is sparse.

For some functions of brains, neurons are hypothesized to be grouped into several layers and process the input in a hierarchical, feedforward fashion. We take humans' vision system as an example and illustrate it in Fig. 1-2. Starting from the retina,
the stimuli sequentially go through different visual areas: LGN, V1, V2, V4 and then IT. The first layer is able to extract some low-level features, such as edges and lines. At subsequent layers, the low-level features are then combined into more complex features. For example, lines are combined into shapes, which are further combined into objects. Therefore, high-level features can be extracted with multiple layers and allow the brain to make high-quality decisions.

Artificial neural networks were designed based on the observations mentioned above. Fig. 1-3 illustrates a type of artificial neural networks, called fully-connected neural network, with only one hidden fully-connected layer. Each layer consists of several neurons, and each neuron is densely connected with all the neurons in the nearby layers (i.e., fully-connected). There is a weight associated with each connection. The neurons in the input layer receive some values and propagate them to the neurons in the middle layer of the network, which is frequently called a hidden layer. The weighted sums from one or more hidden layers are ultimately propagated to the output layer, which presents the final outputs of the network to the user. In mathematics, artificial neural networks have been shown to be a class of universal approximators [12]. We can always find a neural network and a set of weights which can compute the function we want (e.g., the relationship between the input and the output). Finding the correct set of weights is the purpose of training.
Despite the universality of fully-connected neural networks, they suffer from the problem of having a large number of weights. Due to the fully-connected topology, the number of weights grows quadratically with the number of neurons in the successive layers. The large number of weights not only causes a storage problem but also makes the neural network tend to overfit the training data and not generalize well.

Convolutional neural networks (CNNs) [13] were then developed to alleviate the
problem by replacing most of the fully-connected layers with convolutional layers. There are two innovations of convolutional layers. First, each neuron in a convolutional layer is only locally connected to a fixed number of neurons in the nearby layers. Second, neurons in the same layer are grouped, and each group shares the same set of weights. With these two simplifications, the number of weights is significantly reduced and only grows linearly with the number of neurons. Moreover, the computation of a convolutional layer can be structured as convolution, which allows efficient processing and gives the name convolutional layer.

Fig.1-4 illustrates the 2-D convolution operation, which is widely used in image filtering. The weighted sum for each output activation is computed by using only a small neighborhood of input activations (i.e., all weights beyond the neighborhood are set to zero), and where the same set of weights are shared for every output (i.e., the weights are space invariant). By borrowing the terminology from image filtering, the grouped weights and the grouped activations are referred to as filters and feature maps.

The 2-D convolution operation can be generalized to a higher-dimensional convolution (Fig. 1-5), which is used in convolutional layers. In the high-dimensional convolution, the input activations of a layer are structured as a set of 2-D input feature maps (ifmaps), each of which is called a channel. Each channel is convolved with a distinct 2-D filter from the stack of filters, one for each channel; this stack of 2-D filters is often referred to as a single 3-D filter. The results of the convolution at each point are summed across all the channels. In addition, a 1-D bias can be added to the filtering results, but some recent networks [14] remove its usage from parts of the layers. The result of this computation is the output activations that comprise one channel of output feature map (ofmap). Additional 3-D filters can be used on the same input to create additional output channels. Finally, multiple input feature maps may be processed together as a batch to potentially improve reuse of the filter weights.

For example, an input image (feature map) to the first layer of a CNN will contain three channels that correspond to the red, green and blue components of the input image.
Modern Deep CNN: 5 - 1000 Layers

Low-Level Features → Features + Clas

Convolution Non-linearity Normalization Pooling Fully Non-linearity

Optional

Figure 1-6: This figure shows the typical architecture of modern convolutional neural networks on the image classification application [4].

Fig. 1-6 shows the typical architecture of modern convolutional neural networks on the image classification application. The network takes an image as an input and generates the class score of each class. The class with the highest score is the most possible class of the input image. The networks are usually composed of five to a thousand of convolutional layers for extracting features and one to three fully-connected layers in the last few layers for classification. In such networks, each layer extracts successively higher-level features. The features are learned from the data through training, which adjusts the weights to maximize the score of the correct class and minimize the score of the wrong classes. The learnable features solve the problem of the traditional machine learning algorithms that require human experts to handcraft the features. Moreover, it is observed that neural networks can learn better features and achieve higher accuracy when more training data is available (Fig. 1-1). In summary, the learnable features and the scalability on available data make neural networks outperform the traditional machine learning algorithms.

Artificial neural networks have revolutionized and achieved the state-of-the-art performance on various applications, such as computer vision [17-20], speech and language processing [21-24], medical examination [25-31], game play [32, 33], and robotics [34-39]. However, the high accuracy of neural networks is at the cost of high
Table 1.1: The table compares the computational complexity of a traditional algorithm with the CNN alternatives on the image classification application [10].

<table>
<thead>
<tr>
<th>Feature</th>
<th>GOP/Mpixel</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hand-crafted HOG [15]</td>
<td>0.7</td>
<td>1.0x</td>
</tr>
<tr>
<td>CNN-based Features [16]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AlexNet</td>
<td>25.8</td>
<td>36.9x</td>
</tr>
<tr>
<td>VGG</td>
<td>610.3</td>
<td>871.9x</td>
</tr>
</tbody>
</table>

computational complexity. Table 1.1 compares the computational complexity of a traditional algorithm using hand-crafted HOG [15] with the CNN alternatives [16] on the image classification application. CNN-based algorithms require much more computation than the traditional hand-crafted HOG method. The high computational complexity leads to the high operating cost of data centers and also hinders the deployment of neural networks on mobile devices.

1.3 Our Contributions

In this thesis, we aim to design low-complexity neural networks. We propose an automatic algorithm, called NetReduce, that gradually reduces the complexity of a pre-trained network until the target complexity is reached while maximizing the accuracy. For the image classification application on the ImageNet dataset [40], NetReduce outperforms other works by up to 1.68x fewer multiply-accumulate operations (MACs) with 0.9% higher accuracy on a highly compact neural network architecture, MobileNet [7].

The proposed algorithm has the following advantages:

- The algorithm automatically determines the architecture of the simplified network instead of manually choosing it.

- The algorithm jointly considers the complexity reduction and the accuracy impact when simplifying the network to achieve a better accuracy-complexity trade-off.

- The algorithm generates not only a network satisfying the given complexity
constraint but also a sequence of simplified networks to provide the efficient frontier of different accuracy-complexity trade-offs. This is very useful for dynamic neural network selection and further analysis.

- The algorithm is easy to interpret instead of being a black box. This makes hyper-parameter tuning easier and more effective and facilitates the further improvement of the algorithm.
Chapter 2

Overview of Neural Network
Simplification Algorithms

In this chapter, we will give an overview of related neural network simplification algorithms. These algorithms can be roughly grouped into four categories: network pruning, quantization, compact network architecture, and knowledge distillation. The proposed algorithm is most related to network pruning and is complementary to the methods in other categories.

2.1 Network Pruning

Neural networks are usually over-parameterized to make network training easier. Therefore, a large number of the weights in a network are redundant and can be removed (e.g., set to zero). This process is called network pruning. Fig. 2-1 illustrates three different network pruning strategies. They are classified based on the outcome of pruning.

Weight pruning prunes individual weights and usually achieves the highest compression rate (i.e., the number of removed weights divided by the total number of weights). However, the hardware support for the sparse data and the sparse computation is needed to obtain actual speed-up. LeCun et al. [41] and Hassibi et

---

1This chapter contains excerpts of our co-authored paper [4].
Weight Pruning

Filter Pruning

Structured Pruning

Figure 2-1: This figure illustrates three different network pruning strategies, where each $3 \times 3$ block is a filter in a layer. Weight pruning prunes individual weights from the filters, and filter pruning removes entire filters. Structured pruning prunes weights with a particular structure, and filter pruning can be viewed as a special case of structured pruning.

al. [42] remove the weights based on the sensitivity of the final objective function to that weight (i.e., remove the weights with the least sensitivity first). However, the complexity of computing the sensitivity is too high for large networks, so the magnitude-based pruning methods [43] use the magnitude of a weight to approximate its sensitivity; specifically, the small-magnitude weights are removed first. Han et al. [6, 44] apply this idea to recent networks and achieve large network size reduction. They iteratively prune and globally fine-tune the network, and the pruned weights will always be zero after being pruned. Jin et al. [45] and Guo et al. [46] extend the magnitude-based methods to allow the restoration of the pruned weights in the previous iterations, with tightly coupled pruning and global fine-tuning stages, for greater network compression.

Filter pruning removes the entire filters. No special hardware support is needed for filter pruning. Moreover, after a filter is removed, the corresponding feature map can also be removed. This further boosts the performance. Unfortunately, filter pruning tends to have lower compression rates than weight pruning for the same accuracy. Hu et al. [47] propose removing filters that frequently generate zero outputs.
after the ReLU layer in the validation set. Srinivas et al. [48] propose merging similar filters into one. Mariet et al. [49] propose merging filters in the FC layers with similar output activations into one.

Structured pruning removes weights with a particular pattern, and filter pruning can be viewed as its special case. The particular pattern can be determined by considering the architecture of the underlying hardware, which results in more efficient processing. Anwar et al. [50] propose different structures and use particle filter to prune the weights. Yu et al. [51] group the weights based on the SIMD width and prune the weights with the magnitude-based methodology.

Most of the above methods require manually choosing the compression rate for each layer. To solve this problem, MorphNet [8] leverages the sparsifying regularizers to automatically determine the layerwise compression rates. ADC [9] uses reinforcement learning to learn a policy for choosing the compression rates. Although they avoid tuning the layerwise compression rates, tuning the regularization terms and the reinforcement learner can be difficult. The proposed algorithm addresses this problem by making the hyper-parameters intuitive, which leads to easier tuning. Moreover, the above methods can be used as the building blocks of the proposed algorithm.

2.2 Quantization

Quantization involves mapping data to a smaller set of quantization levels. The ultimate goal is to minimize the error between the reconstructed data from the quantization levels and the original data. The number of quantization levels reflects the precision and ultimately the number of bits (i.e., the bitwidth) required to represent the data (usually $\log_2$ of the number of levels). The benefits of quantization include reduced storage cost and reduced computation requirements.

Quantization methods can be classified based on the spacing of the quantization levels:

Uniform quantization uses the levels that are uniformly spaced out (Fig. 2-2(a)). Without any fine-tuning of the weights, the bitwidth can be reduced to 8 bits
Figure 2-2: This figure illustrates various methods of quantization [5, 6].
for the weights and 10 bits for the activations [52]; with fine-tuning, both weights and activations can reach 8-bits [53]. BinaryConnect (BC) [54] introduces the concept of binary weights (i.e., -1 and 1), where using a binary weight reduces the multiplication in the MAC to addition and subtraction only. This is later extended in Binarized Neural Networks (BNN) [55] that use binary weights and activations, which reduces the MAC to an XNOR operation. However, BC and BNN have an accuracy loss of 19% and 29.8% on AlexNet [56] with the ImageNet dataset [40], respectively [57]. In order to reduce this accuracy loss, Binary Weight Nets (BWN) and XNOR-Nets introduce multiplying the outputs with a scale factor to recover the dynamic range. BWN reduces the accuracy loss to 0.8%, while XNOR-Nets reduces the loss to 11%. The loss of XNOR-Net can be further reduced by increasing the precision of the activations to be slightly larger than one bit. For instance, HWGQ-Net [58] allows the activations to have 2-bits, while the weights remain at 1-bit, and reduces the accuracy loss to 5.2%.

**Nonuniform Quantization** uses the levels that are nonuniformly spaced out. Lee et al. [5] assign the quantization levels based on a logarithmic distribution as shown in Fig 2-2(b). The weights and activations are more equally distributed across the different levels, and each level is used more efficiently, which results in less quantization error. Incremental Network Quantization (INQ) can be used to further reduce the loss in accuracy by dividing the large and small weights into different groups, and then iteratively quantizing and re-training the weights [59]. Deep compression [6] proposes weight sharing to force several weights to share a single value. This reduces the number of unique weights in a filter or a layer, as shown in Fig. 2-2(c).

### 2.3 Compact Network Architecture

Neural networks can also be simplified by improving the network architecture itself. The trend is to replace a large filter with a series of smaller filters, which have fewer weights in total; when the filters are applied sequentially, they achieve the same overall effective receptive field (i.e., the region the filter uses from the input image to
compute an output). This approach can be applied during the network architecture design (before training) or by decomposing the filters of a pre-trained network (after training).

**Before Training:** In recent neural networks, filters with a smaller width and height are used more frequently because concatenating several of them can emulate a larger filter. Simonyan et al. [60] propose replacing one $5 \times 5$ convolution with two $3 \times 3$ convolutions (Fig. 2-3(a)). Szegedy el al. [61] show that one $M \times M$ convolution can be decomposed into two 1-D convolutions, one $1 \times M$ and one $M \times 1$ convolution (Fig. 2-3(b)). Xception [63] and MobileNets [7] adopt depthwise decomposition, which replaces a 3-D $M \times M$ convolution by a set of 2-D $M \times M$ convolutions (i.e., applied
Figure 2-4: Knowledge distillation matches the class scores of a small neural network (student) to that of a large neural network or a network ensemble (teacher).

After Training: Tensor decomposition can be used to decompose filters in a pre-trained neural network without impacting the accuracy. It treats weights in a layer as a 4-D tensor and breaks it into a combination of smaller tensors (i.e., several layers). The low-rank approximation can then be applied to further increase the compression rate at the cost of accuracy degradation, which can be restored by fine-tuning the weights. This approach is demonstrated using Canonical Polyadic (CP) decomposition, a high-order extension of singular value decomposition that can be solved by various methods, such as a greedy algorithm [64] or a non-linear least-square method [65]. Combining CP-decomposition with low-rank approximation achieves a 4.5× speed-up on CPUs [65]. However, CP-decomposition cannot be computed in a numerically stable way when the dimension of the tensor, which represents the weights, is larger than two [65]. To alleviate this problem, Tucker decomposition is adopted instead in [66].
2.4 Knowledge Distillation

Knowledge distillation transfers the knowledge learned by a complex model (teacher) to a simpler model (student). The student network can therefore achieve an accuracy that would be unachievable if it was directly trained with the same dataset from scratch [67, 68]. Bucilua et al. [67] propose matching the class scores of the student network to that of the teacher network, as shown in Fig. 2-4. Hinton et al. [69] introduce a configurable softmax layer, which can generate softer class probabilities where the smaller values retain more information, and match the class probabilities of the student network to the teacher network. Romero et al. [70] incorporate the intermediate feature maps of the teacher network as the extra hints to train the student network.
Chapter 3

The Proposed Progressive Barrier Based Algorithm for Simplifying Neural Networks

In this chapter, we propose an algorithm, called NetReduce, that simplifies a pre-trained neural network to meet a given complexity constraint while maximizing the accuracy. By using progressive barriers, NetReduce divides the simplification process into a series of simpler steps and simplifies the input network iteratively. This design provides automatic efficient architecture discovery, better accuracy-complexity trade-offs, efficient frontier generation, and higher interpretability.

The following sections are organized as follows: Sec. 3.1 introduces the problem formulation and discusses how we use a progressive barrier based approach to solve the network simplification problem. We give an overview of the algorithm in Sec. 3.2. Sec. 3.3 explains the details of the algorithm.
3.1 Problem Formulation

NetReduce aims to solve the following non-convex constrained problem:

$$\begin{align*}
\text{maximize} \quad & Acc(Net) \\
\text{subject to} \quad & Comp(Net) \leq Cons,
\end{align*}$$

where $Net$ is a simplified network from the initial pre-trained network, $Acc(\cdot)$ evaluates the accuracy, $Comp(\cdot)$ calculates the computational complexity, and $Cons$ is the constraint on the computational complexity. This problem is hard to solve due to its high non-linearity. For example, the relationship between a network architecture and its accuracy is non-linear.

Based on an idea similar to the progressive barrier approach [71], which is a derivative-free optimization method, NetReduce breaks this problem into the following series of easier problems and solves it iteratively:

$$\begin{align*}
\text{maximize} \quad & Acc(Net_i) \\
\text{subject to} \quad & Comp(Net_i) \leq Comp(Net_{i-1}) - \Delta C_i,
\end{align*}$$

where $Net_i$ is the network generated by the $i^{th}$ iteration and $Net_0$ is the initial pre-trained network. $\Delta C_i$, which is larger than zero, indicates how much the constraint tightens in the $i^{th}$ iteration. As the number of iterations increases, the constraint (i.e., $Comp(Net_{i-1}) - \Delta C_i$) gradually becomes tighter. If $\Delta C_i$ is small enough, the constraint can be easily satisfied by slightly simplifying a single layer. Moreover, $\Delta C_i$ can vary from iteration to iteration. This is referred to as complexity reduction scheduling, which is similar to the concept of learning rate scheduling. The scheduling controls the required number of iterations and the quality of the simplified networks.

The algorithm terminates when $Comp(Net_{i-1}) - \Delta C_i$ is equal to or smaller than $Cons$. It outputs the final network satisfying the constraint and can also generate a sequence of simplified networks (i.e., the highest accuracy network from each iteration $Net_1, ..., Net_i$) to provide the efficient frontier of accuracy-complexity trade-offs.
Algorithm 1: NetReduce

Input: Pretrained Network: $Net_0$ (with $K$ CONV and FC layers), Constraint: $Cons$, Complexity Reduction Schedule: $\Delta C_i$

Output: Simplified Network Satisfying the Constraint: $\hat{Net}$

1. $i = 0$
2. $Comp_i = ComputeComplexity(Net_i)$
3. while $Comp_i > Cons$ do
   4. $Cons = Comp_i - \Delta C_i$
   5. for $k$ from 1 to $K$ do
      /* ComputeComplexity is also called inside ChooseNumFilters for choosing the correct number of filters that satisfies the constraint. */
      6. $N_{Filt_k}, Comp_{Simp_k} = ChooseNumFilters(Net_i, k, Cons)$
      7. $Net_{Simp_k} = ChooseWhichFilters(Net_i, k, N_{Filt_k})$
      8. $Net_{Simp_k} = ShortTermFineTune(Net_{Simp_k})$
      9. $Net_{i+1}, Comp_{i+1} = PickHighestAccuracy(Net_{Simp}, Comp_{Simp})$
   10. $i = i + 1$
11. $\hat{Net} = LongTermFineTune(Net_i)$
12. return $\hat{Net}$

3.2 Algorithm Overview

The NetAdapt algorithm is detailed in pseudo code in Algorithm 1 and in Fig. 3-1. Each iteration solves Eq. 3.1 by reducing the number of filters in a single CONV or FC layer\(^1\) (the Choose # of Filters and Choose Which Filters blocks in Fig. 3-1). The number of filters to remove from a layer is guided by the constraint of that iteration. The simplified network is then fine-tuned for a short length of time in order to restore some accuracy (the Short-Term Fine-Tune block in Fig. 3-1).

In each iteration, the first two steps (Choose # of Filters and Choose Which Filters blocks in Fig. 3-1) are applied on each of the CONV or FC layers individually. As a result, NetAdapt generates $K$ (i.e., the number of CONV and FC layers) network candidates in one iteration, each of which has a single layer simplified from the previous iteration. After being fine-tuned, the proposed network candidate with the highest accuracy is carried over to the next iteration (the Pick Highest Accu-

\(^1\)The algorithm can also be applied to a group of multiple layers as a single unit (instead of a single layer). For example, in ResNet [14], we can treat a residual block as a single unit to speed up the simplification process.
Figure 3-1: This figure visualizes the algorithm flow of NetReduce. At each iteration, NetReduce reduces the complexity by simplifying (i.e., removing filters from) one layer. In order to maximize accuracy, it tries to simplify each layer individually and picks the simplified network that has the highest accuracy. Once the target complexity is met, the chosen network is then fine-tuned again until convergence.

Note that the selection of the network for the next iteration should not be made based on the test set to avoid overfitted results. Finally, once the target complexity is met, the chosen network is fine-tuned again until convergence (the Long-Term Fine-Tune block in Fig. 3-1).

3.3 Algorithm Details

This section describes the key blocks in the NetReduce algorithm (Fig. 3-1).
3.3.1 Choose Number of Filters

This step focuses on determining how many filters to preserve in a specific layer based on the current constraint. NetReduce gradually reduces the number of filters in the target layer and computes the complexity of each of the simplified networks. The maximum number of filters that can satisfy the current constraint will be chosen. It should be noted that when some filters are removed from a layer, the associated channels in the following layers should also be removed. Therefore, the change in the complexity of other layers needs to be factored in.

3.3.2 Choose Which Filters

This step is responsible for choosing which filters to preserve based on the architecture from the previous step. NetAdapt removes entire filters instead of individual weights. There are two advantages. First, most of the existing platforms cannot efficiently utilize sparse filters but can fully take the benefit of removing entire filters. Second, after a filter is removed, the corresponding feature map can also be removed. Because I/O operations for reading/writing the feature maps are usually much more expensive than doing computations, reducing feature maps give an extra performance boost.

NetReduce uses the magnitude-based method to keep the algorithm simple. The \( N \) filters that have the largest \( \ell_2 \)-norm on their weights will be kept, where \( N \) is the number of filters determined by the previous step. Other methods introduced in Sec. 2.1 can potentially be adopted in this step.

3.3.3 Short-/Long-Term Fine-Tune

Both the short-term and long-term fine-tuning steps in NetReduce involve network-wise end-to-end fine-tuning. The only difference is that they fine-tune the network with different numbers of iterations.

At each iteration of the algorithm, NetReduce fine-tunes the simplified networks with a relatively smaller number of iterations (i.e., short-term) to regain accuracy. This step can be carried out in parallel or in sequence. As shown in Sec. 4.3.2, without
short-term fine-tuning, the accuracy will drop to zero in a few iterations, and this misleads the algorithm to choose the wrong network candidate. Because NetReduce starts from a pre-trained network and only slightly simplifies a layer in each iteration, the accuracy can be rapidly restored, and a small number of fine-tuning iterations are enough.

As the algorithm proceeds, the network is continuously trained but does not converge. Once the final simplified network is obtained, NetReduce fine-tunes the network with more iterations until convergence (i.e., long-term) as the final step. The impact of this step is analyzed in Sec. 4.3.3.
Chapter 4

Experiment Results

In this chapter, we apply the proposed NetReduce algorithm to MobileNets [7], which are designed for mobile applications, and experiment on the ImageNet dataset [40]. We did not apply NetReduce on larger networks like ResNet [14] and VGG [60] because these networks are seldom deployed on mobile platforms; it is also more difficult to simplify an already efficient network such as MobileNets than larger networks. We benchmark NetReduce against three state-of-the-art network simplification methods:

- **MobileNet Family** [7]\(^1\) consists of networks that are simplified from a reference MobileNet architecture (Sec. 2.3) by using two simple but effective hyperparameters, the width multiplier and the resolution multiplier. Width multiplier scales the number of filters by a percentage across all convolutional (CONV) and fully-connected (FC) layers, and resolution multiplier scales the resolution of the input image. The two multipliers can be used together. We use the notation “50% MobileNet (128)” to denote applying the width multiplier of 50% on MobileNet with the input image resolution of 128.

- **MorphNet** [8] is an automatic network simplification algorithm based on sparsifying regularization.

\(^1\)For the MobileNet family, we report the accuracy numbers of our trained version. Except for one network whose accuracy is decreased by 0.2%, all other networks achieve the same or higher accuracy compared to the numbers reported in [7].
• ADC [9] is an automatic network simplification algorithm based on reinforcement learning.

To demonstrate the effectiveness of NetReduce, we simplified the largest MobileNet (100% MobileNet (224)) and a much smaller MobileNet (50% MobileNet (128)). We also performed ablation studies to investigate the effectiveness and impact of several components of NetReduce.

4.1 Detailed Settings

4.1.1 Algorithm Configuration

MobileNets [7] are based on the depthwise decomposition, which factorizes a M×M 3-D convolutions into a set of M×M 2-D convolutions (i.e., depthwise layer) and a set of 1×1 3-D convolutions (i.e., pointwise layer). In the experiments, we simplified each depthwise layer with the corresponding pointwise layer and chose which filters to keep based on the pointwise layer. When simplifying 50% MobileNet (128), the initial complexity reduction ($\Delta C_0$ in Eq. 3.1) at the first iteration is around 1.5M, and it decays at the rate of 0.96 per iteration. When simplifying 100% MobileNet (224), the initial complexity reduction at the first iteration is around 16.8M, and it decays at the rate of 0.96 per iteration.

4.1.2 Training Configuration

To avoid overfitting to the test set (in this case, the ImageNet validation set), we preserved ten thousand images from the training set, ten images per class, as the holdout set. The new training set without the holdout images was used to perform short-term fine-tuning, and the holdout set was used to pick the highest accuracy network out of the network candidates at each iteration. The whole training set is used for the long-term fine-tuning, which is performed once in the last step of NetReduce.
Figure 4-1: The figure compares NetReduce (simplifying 50% MobileNet (128)) with the MobileNet family [7] and MorphNet [8].

<table>
<thead>
<tr>
<th>Network</th>
<th>Top-1 Accuracy (%)</th>
<th># of MACs ($\times 10^6$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>25% MobileNet (224) [7]</td>
<td>52.8 (+0)</td>
<td>41.0 (100%)</td>
</tr>
<tr>
<td>NetReduce</td>
<td>53.7 (+0.9)</td>
<td>24.4 (59%)</td>
</tr>
<tr>
<td>MorphNet [8]</td>
<td>46.0 (+0)</td>
<td>15.0 (100%)</td>
</tr>
<tr>
<td>NetReduce</td>
<td>46.3 (+0.3)</td>
<td>11.0 (73%)</td>
</tr>
</tbody>
</table>

Table 4.1: A pairwise comparison between NetReduce (simplifying 50% MobileNet (128)) and the other two benchmark algorithms, the MobileNet family [7] and MorphNet [8]. We compare the number of MACs at similar accuracy.

The batch size is 128, and we use 16 GPUs to train a network with asynchronous parameter update. The optimizer is the RMSProp [72] with both momentum and decay set to 0.9. We used 0.045 as the initial learning rate for the long-term fine-tuning and 0.0045 for the short-term fine-tuning. The learning rate decays 0.98 per epoch. The short-term fine-tuning uses 40K iterations. All accuracy numbers are reported on the validation set to show the actual performance.

4.2 Comparison with Benchmark Algorithms

4.2.1 Simplifying 50% MobileNet (128)

In this experiment, we apply NetReduce to simplify 50% MobileNet (128). 50% MobileNet (128) is one of the most compact networks and much harder to simplify than other larger networks. The results are summarized and compared with the MobileNet family [7] and MorphNet [8] in Fig. 4-1 and Table 4.1. We observe that...
Figure 4-2: The figure compares NetReduce (simplifying 100% MobileNet (224)) with the MobileNet family [7] and ADC [9].

<table>
<thead>
<tr>
<th>Network</th>
<th>Top-1 Accuracy (%)</th>
<th># of MACs ($\times 10^6$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100% MobileNet (192) [7]</td>
<td>69.3 (+0)</td>
<td>418.1 (100%)</td>
</tr>
<tr>
<td>NetReduce</td>
<td>69.4 (+0.1)</td>
<td>309.5 (74%)</td>
</tr>
<tr>
<td>ADC [9]</td>
<td>68.8 (+0)</td>
<td>304.2 (100%)</td>
</tr>
<tr>
<td>NetReduce</td>
<td>68.8 (+0.0)</td>
<td>253.5 (83%)</td>
</tr>
</tbody>
</table>

Table 4.2: A pairwise comparison between NetReduce (simplifying 100% MobileNet (224)) and the other two benchmark algorithms, the MobileNet family [7] and ADC [9]. We compare the number of MACs at similar accuracy.

NetReduce outperforms the MobileNet family by up to 1.68x fewer MACs with 0.9% higher accuracy. For MorphNet, NetReduce’s result has 1.36x fewer MACs with 0.3% higher accuracy.

4.2.2 Simplifying 100% MobileNet (224)

In this experiment, we apply NetReduce to simplify 100% MobileNet (224). 100% MobileNet (224) is the largest MobileNet and achieves the highest accuracy. Because its number of MACs is approximately 11.6x higher than that of 50% MobileNet (128), we scale the initial complexity reduction by around 11.6x. The results are shown and compared with the MobileNet family [7] and ADC [9] in Fig. 4-2 and Table 4.2. NetReduce achieves higher accuracy than the MobileNet family and ADC while decreasing the number of MACs by up to 1.35x and 1.20x, respectively.
Figure 4-3: This figure shows the sensitivity of simplifying each layer of 50% MobileNet (128) to give an around 6.8% drop in the number of MACs. The simplified networks are fine-tuned for 150 epochs.

4.3 Ablation Studies

In this section, we study the effectiveness and impact of several components of NetReduce. We first show that the accuracy drop when simplifying different layers can vary significantly. Then the short-term and long-term fine-tuning are analyzed. Finally, we compare the architectures of the simplified networks with the MobileNet family to provide insights.

4.3.1 Layer Sensitivity to Simplification

Fig. 4-3 provides one example of layer sensitivity to simplification. In this experiment, we start from 50% MobileNet (128) and simplify each layer to achieve an around 6.8% drop in the number of MACs. The simplified networks are then fine-tuned for 150 epochs.

The result shows that different layers have different sensitivity to simplification. The first two and the last three convolutional layers are more susceptible than other layers. More specifically, the last convolutional layer (layer 13) is the most sensitive and the layer 4 is the least sensitive. The difference is 2.7%. If we reduce the number of MACs more, the difference in accuracy between two different sets of compression rates can be further increased.
4.3.2 Impact of Short-Term Fine-Tuning

Fig. 4-4 illustrates the necessity of incorporating the short-term fine-tuning into NetReduce. Without performing short-term fine-tuning, the accuracy rapidly drops to nearly zero. In this low accuracy region, the accuracy numbers are like random numbers, and the order of them does not provide any meaningful guidance. This misleads the algorithm to pick a wrong architecture and hence gives a low accuracy.

Unlike training a network from scratch, where the weights are initialized randomly, NetReduce starts from a better set of weights so that a small number of fine-tuning iterations are enough to significantly restore the accuracy. In this example, after fine-tuning the simplified network for 40K iterations, the accuracy can always be kept above 38% even though the networks have not yet converged. The restored accuracy allows the algorithm to make a better decision. Empirically, further increasing the number of iterations improves the accuracy. We found that using 40K iterations leads to a good accuracy-speed trade-off.

4.3.3 Impact of Long-Term Fine-Tuning

Fig. 4-5 illustrates the importance of performing the long-term fine-tuning. Although the short-term fine-tuning restores the accuracy well, the long-term fine-tuning can
Figure 4-5: The comparison between with and without performing long-term fine-tuning (LFT) when simplifying 50% MobileNet (128). Although the short-term fine-tuning preserves the accuracy well, the long-term fine-tuning gives the extra 3% in average (from 1.9% to 4.5%).

still increase the accuracy by up to another 4.5% or 3% in average. Because of the short training time of the short-term fine-tuning, the fine-tuned network has not yet converged. Therefore, it is not surprising that the final long-term fine-tuning step can further increase the accuracy.

Moreover, we can observe that the long-term fine-tuning becomes more important as the network simplification goes. The reason is that the accuracy of small networks usually increases slower than large networks during training, so the short-term fine-tuning leaves more room for the long-term fine-tuning to improve.

### 4.4 Analysis of Simplified Network Architectures

Fig. 4-6 and Fig. 4-7 compare the simplified networks with the corresponding networks in the MobileNet family with similar numbers of MACs (but higher accuracy). There are two main observations.

First, there are some bottleneck layers that have fewer filters than the neighboring layers, such as the layer 7, 9 and 10 in the simplified 50% MobileNet (128) and the layer 9 in the simplified 100% MobileNet (224). The bottleneck layers are used in some hand-crafted high-performance networks to maintain the accuracy while reducing the
computational complexity, such as [14].

Second, the simplified networks keep more filters in the last few layers than the corresponding MobileNet family. One possible explanation is that the ImageNet dataset contains one thousand classes, so more feature maps are needed by the last FC layer to do correct classification. Moreover, there is a global pooling layer right before the fully-connected layer that potentially causes huge information loss. Therefore, more filters in the last few layers may be needed for making compensation.
Chapter 5

Conclusion

In summary, we propose an automated algorithm that simplifies a pre-trained neural network to reduce its computational complexity. The algorithm first loosens the constraint on the complexity and then gradually tightens it until the target complexity is met. During this process, a family of simplified neural networks with different accuracy-complexity trade-offs are generated. Moreover, the algorithm jointly considers the complexity reduction and the accuracy drop, and the hyper-parameters are also more intuitive and easier to tune when compared with other automated algorithms for simplifying neural networks. These features enable designing neural networks with a better accuracy-complexity trade-off, which facilitates the deployment of neural networks on more applications.

The future work will be incorporating hardware-dependent metrics, such as energy and latency, into the proposed algorithm. These metrics consider more factors (e.g., the toolchain) than the number of MACs. Therefore, simplifying a neural network based on hardware-dependent metrics may lead to better performance.
Bibliography


[22] Li Deng, Jinyu Li, Jui-Ting Huang, Kaisheng Yao, Dong Yu, Frank Seide, Michael Seltzer, Geoff Zweig, Xiaodong He, Jason Williams, et al. Recent advances in deep learning for speech research at Microsoft. In International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2013.


[39] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. Safe, multi-agent, reinforcement learning for autonomous driving. In NIPS Workshop on Learning, Inference and Control of Multi-Agent Systems, 2016.


[52] Yufei Ma, Naveen Suda, Yu Cao, Jae-Sun Seo, and Sarma Vrudhula. Scalable and modularized RTL compilation of Convolutional Neural Networks onto FPGA. In International Conference on Field-Programmable Logic and Applications (FPL), 2016.


[64] Emily Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting Linear Structure Within Convolutional Networks for Efficient Evalu-


