UNIVERSITY OF CALIFORNIA
RIVERSIDE

Adaptive Solution to Compress Deep Neural Networks for Resource-Constrained Devices

A Thesis submitted in partial satisfaction
of the requirements for the degree of

Master of Science

in

Electrical Engineering

by

Ruzhuo Wang

March 2019

Thesis Committee:

Professor Hyoseung Kim, Chairperson
Professor Shaolei Ren,
Professor Daniel Wong

The Thesis of Ruzhuo Wang is approved:

Committee Chairperson

University of California, Riverside

Acknowledgments

First and foremost, I would like to thank Professor Hyoseung Kim for his expert advice and encouragement helping me finish this thesis, as well as my colleagues in the Rten lab for their kindly help, they supported me greatly and were always willing to help me.

To my parents and my young sister: for always being there to support me when I was studying at the United States these two years. Especially my father, who always gave me guides when I was confused about something here. To my mother, although I may not plan to pursue PhD right now, I tried my best to not live up to you. To my little sister: Little buddy you are smarter than me, wish you can have a health and happy life in the future.

To my parents for all the support.

ABSTRACT OF THE THESIS

Adaptive Solution to Compress Deep Neural Networks for Resource-Constrained Devices

by

Ruzhuo Wang

Master of Science, Graduate Program in Electrical Engineering
University of California, Riverside, March 2019
Professor Hyoseung Kim, Chairperson

Recent advantages of deep neural networks (DNNs) motivate their use in many applications, but in order to apply DNNs on resource-constrained devices like embedded systems, there are many difficulties to overcome. One effective solution is compressing DNNs. However, existing compression techniques have problems such as, they can only compress specific types of neural networks or they have to find the sparsity of weight matrices. This motivates us to develop compression methods for pre-trained deep neural networks in order to meet specific requirements, such as reducing execution time and decreasing model size.

We propose a new compression solution, called Adaptive-Surgery, which has two important properties. First, Adaptive-Surgery presents a unified approach that is able to compress all commonly used deep learning structures, including fully-connected and convolutional neural networks, as well as their combinations. It does so by applying different compression methods based on the type of the structures. Second, Adaptive-Surgery targets at the weights matrices and compresses the weights matrices by reserving only the most significant parameters while trying not to lose too much accuracy of the original deep

neural networks. Importantly, unlike the traditional dropout model compression method that randomly drops components in the weights matrices, Adaptive-Surgery will use singular value decomposition in the process of pruning fully-connected structures. We call this new compression method SVD-based dropout.

After compressed by Adaptive-Surgery, the new model can be directly used on embedded systems without further modifications. In our evaluation, Adaptive-Surgery is used to compress two different deep neural networks, and we will test the performance of generated models on Raspberry Pi 3. Each deep neural network can be compressed in three different degrees: 'Rare', 'Medium' and 'Well-Done', where the corresponding compression ratios (ratio of the size of pruned parameters to that of the original parameters) are 43.75%, 75% and 93.75%, respectively. Experiment results show that our proposed work can yield a significant reduce in execution time and model size without causing appreciable loss in accuracy.(e.g., Adaptive-Surgery can have the compression ratio of 75% on the modified Alexnet, while the accuracy is decreased only from 90.6% to 90.0%)

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Because there is massive amount of data that is available nowadays and has been gathered over the last year and decades, deep learning has best-in-class performance on solving problems with huge amount of data that significantly outperforms other approaches in plenty of domains. This enables neural networks to really show their potential since they get smarter the more data you used to train them. Much researches in recent years have focused on remarkable potential of neural networks, and numerous experiments have be established to apply neural networks in multiple domains such as a deep cascaded multi-task framework for face detection [9], a deep convolutional neural network to classify the 1.2 million high-resolution images [1] and a convolutional neural network (CNN) to map raw pixels from a single front-facing camera directly to steering command [2]. More further, in order to make it possible for people to take the advantages of neural networks in daily life, implementations of neural networks on embedded systems such as running deep learning

models locally on the wearable devices [10] or building a mobile audio sensing framework from deep neural networks [11] play a pivot role.

However, currently there exists a big challenge for the growing need for computational ability to deal with huge amount of data and the ability of resource constrained embedded systems to execute deep neural networks [6]. In the process of seeking a way to narrow the gap, researchers find a powerful tool called model compression including pruning the fully-connected structure using singular value decomposition [3] or dropout [4] and pruning the feature map by magnitude of kernel weights in convolutional structure [7].

In 2017, a new compression solution called DeepIoT [7] is created, it presents a striking approach that compresses most deep learning structures for sensing applications, including fully-connected, convolutional, and recurrent neural networks, as well as their combinations. It can also reduce the size of deep neural networks by over 90% without loss of accuracy. It does so by using a recurrent neural network to learn the parameters redundancies of dense matrices layer by layer. But it takes massive time to learn the redundancies and we do not see the experiment results for DeepIoT that show its ability to manually set the compression ratio for itself.

We want to keep the advantages of this framework such as the ability to compress all commonly used deep learning structures and solve the shortage of this framework such as unable to manually set compression ratio, thus we develop a new compression solution named Adaptive-Surgery. Adaptive-Surgery can have a unified approach for compressing the convolutional structure, fully-connected structure as well as their combinations. More-

over, we can set three compression ratios for Adaptive-Surgery which are 43.75%, 75% and 93.75%.

1.2 Thesis Outline

The thesis is organised as follows:

In chapter 2, we first discuss the technical details about: Singular Value Decomposition (SVD) for compressing fully-connected structure; Dropout for compressing fully-connected structure, because it helps to understand the new compression method we build called SVD-based dropout. We will discuss the DeepIoT [7] at the end of chapter two, so that it will be more clear to see the differences between DeepIoT and Adaptive-Surgery.

In chapter 3, we begin with presenting the system framework of our thesis by providing the technical details of Adaptive-Surgery framework, such as what rules it will obey to compress different DNN structures. Next we present the mathematical formula for SVD-based dropout, and we present a flow chart to illustrate the compression process of SVD-based dropout. Finally we manage to demonstrate the compression methods Adaptive-Surgery used to compressing convolutional structure.

In chapter 4, we first introduce two DNN models, one is modified Alexnet, the other one is Cifar10-quick. We conduct experiments on these two DNN models, so it will be helpful to understand our work by having an overall idea of this two DNN models. Then we present the experiment environment and experiment results to evaluate the performance of Adaptive-Surgery. We will show the process of how we conduct our experiments on two

different DNN models under two different criterion. Finally, we will summarize observations of the experiments and evaluate the performance of Adaptive-Surgery.

In chapter 5, we revisit the previous researches and recall the specific limitations for previous works. Then we summarize the methodologies of Adaptive-Surgery and the observations from experiments, as well as our contributions for this specific area. And we discuss the future work for our works.

1.3 Contribution

The contributions of this thesis are summarized as follows:

- We proposed the Adaptive-Surgery framework, which keeps the advantages of the previous works and can take a input compression ratio from user without causing unacceptable accuracy penalty in the compressing process.
- We make efforts to combine two commonly-used compression methods(singular value decomposition and dropout) to a new compression method called SVD-based dropout that can be applied on fully-connected structure.
- We propose the Adaptive-Surgery framework, which uses the combination of different compression methods to have a unified compression approach for the whole DNN model.

Chapter 2

Background and Related work

In this chapter, we want to provide readers with a basic idea of some commonly-used model compression methods as well as a good compressor-critic framework. We begin with introducing three model compression methodologies which compress the original DNN models by parameter pruning or weight reconstruction. Then we discuss the detail of the compressor-critic framework DeepIoT as well as the limitation for their work.

2.1 Model compression methods

2.1.1 Least weight prune

Before we introduce least weight prune model compression method, we need first simply introduce the Singular Value Decomposition(SVD) [15]. For SVD, it has three important properties:

- It can be applied on any $m \times n$ matrices.

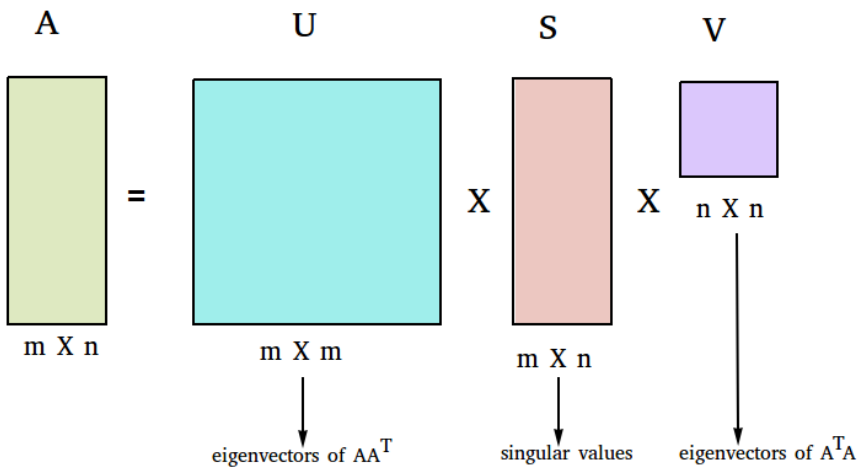


Figure 2.1: Singular Value Decomposition [16]

- The mathematical formula is:

$$A_{m\times n} = U_{m\times m} \cdot S_{m\times n} \cdot V_{n\times n}$$

- The Diagonal value of S matrix are the non-negative square roots of the eigenvalues of $A^T A$, they are called singular values.

The least weight prune [14] compression method targets at the convolutional layer in the DNN model. Each convolutional layer consists of many feature maps, which is also called convolutional kernel, these feature maps are responsible for the computation time in the forward and backward propagation in the DNN model. This method will prune the less useful filters from a pre-trained model for accelerating the execution while minimizing the accuracy penalty. They first calculate the l_1-norm of a convolutional kernel in each layer, this value also gives an expectation of the magnitude of the output feature map. Filters

with smaller kernel weights tend to have smaller influence on the forward and backward propagation process, so we will prune the filters with small kernel weights.

2.1.2 Weight reconstruction by Singular Value Decomposition

Weight reconstruction by Singular Value Decomposition [3] targets at two fully-connected layers. For two fully-connected layers L and $L+1$, updating states of all nodes requires evaluating the product: $W^L \cdot x^L$, where, $x^L \in R^n$ is the state of nodes in the previous layer and $W^L \in R^{m \times n}$ is the matrix representing all the connections between layer L and $L+1$. Now, in order to compress the original DNN model by decrease the matrix multiplication, the basic idea is to replace the weight matrix W^L with a product of two different matrices, i.e.,

$$W^L = U \cdot V$$

Under SVD, the weight matrix can be efficiently factorized as:

$$W^L_{m \times n} = X_{m \times m} \cdot \Sigma_{m \times n} \cdot N^T_{n \times n}$$

where, $\Sigma_{m \times n}$ is a rectangular diagonal matrix containing L *singular values* of $W^L_{m \times n}$ as the diagonal elements. To gain computational efficiency the weight matrix can be approximated well by keeping k highest singular values, meanwhile, we need to prune the X matrix and N^T matrix correspondingly, i.e.:

$$W^L_{m \times n} \approx X_{m \times k} \cdot \Sigma_{k \times k} \cdot N^T_{k \times n}$$

Now, the architecture of a fully-connected layer of a DNN model can be modified by replacing W^L with $U = X_{m \times k}$ and $V = \Sigma_{k \times k} \cdot N^T_{k \times n}$, as is shown in the Figure.

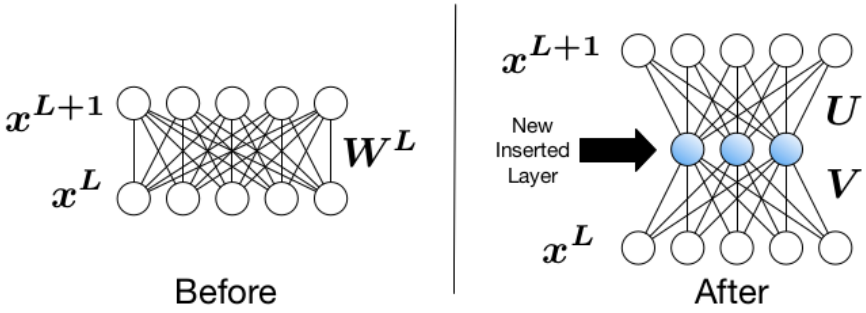
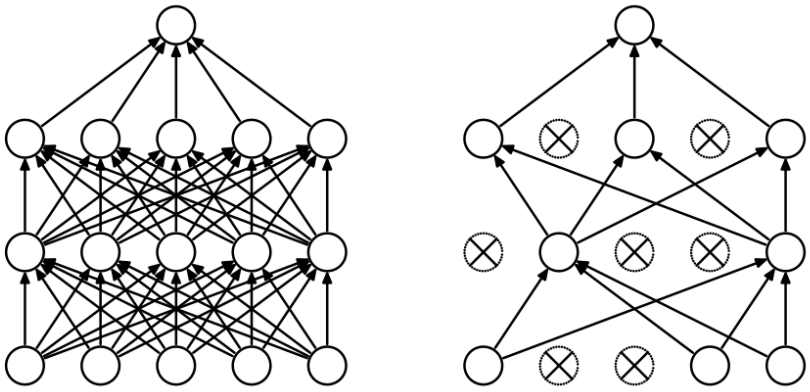


Figure 2.2: Layer insertion for SVD decomposition [3]

2.1.3 Dropout

For Dropout method, it target at the fully-connected structure in the DNN models. In [4], the dropout method is described as "The term 'dropout' refers to dropping out units (hidden and visible) in a neural network. By dropping a unit out, it means temporarily removing it from the network, along with all its incoming and outgoing connections.". In the training process the neurons in each fully-connected layers are dropped randomly, so that there are plenty of thinner DNN models are generated in order to prevent overfitting. But the dropped neurons will come back in the testing process, which means the matrix computation in the testing process does not change.



(a) Standard Neural Net (b) After applying dropout.

Figure 2.3: Dropout Neural Net Model. $Left$: A standard neural net with 2 hidden layers. $Right$: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped. [4]

2.2 Model compression framework

2.2.1 DeepIoT

There is a paper that introduces a compressor-critic framework called $DeepIoT$ [7]. This framework obtains the optimal dropout probabilities for the neural network and exploits the network parameters themselves. In fully-connected neural networks, neurons are dropped in each layer; in convolutional neural networks, filters are dropped in each layer. This means that DeepIoT can be applied to all commonly-used neural network structures and their combinations. DeepIoT use a recurrent neural network to learn the parameter redundancies, and generates the dropout probabilities layer by layer.

DeepIoT reduces the size of deep neural networks by 90% to 98.9%. It is thus able to shorten execution time by 71.4% to 94.5%, and decrease energy consumption by 72.2% to 95.7%. These improvements are achieved without loss of accuracy. In order to have no

loss on the accuracy, it will be very cautious about the compression process, which means it will compress the network little by little. Meanwhile, for each compression step, it needs a recurrent neural network to learn the redundancies and compress the neural network based on this redundancies information, then retrains the compressed networks. This means it will take huge time for each compression step and the time consuming works accumulate through the whole compression process. We want to find a solution that compress the neural network that costs relatively less time and do not have too much penalty on the accuracy, meanwhile, DeepIoT does not present the ability to take a compression ratio as input. So we design Adaptive-Surgery that can compress the commonly used neural network to three user input compression ratios with acceptable compression time and acceptable accuracy penalty.

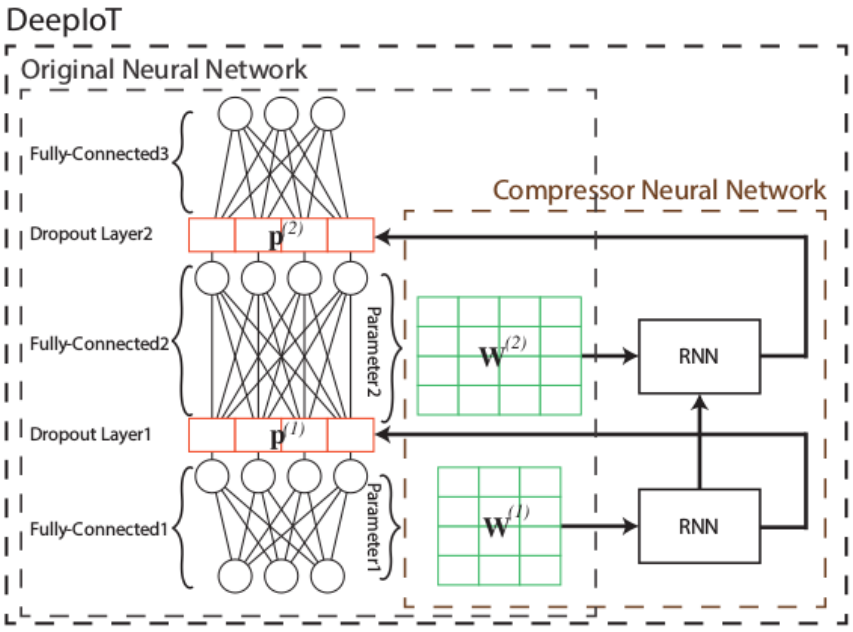


Figure 2.4: Overall DeepIoT system framework. Orange boxes represent dropout operations. Green boxes represent the parameters of the original neural network (this figure is captured from [7]).

Chapter 3

System Framework

In this chapter, we will first introduce the framework we propose named Adaptive-Surgery, and exactly what compression methods it will apply on different structures. Then we will present a new model compression method for fully-connected structure, which is SVD-based dropout. Finally, we introduce how we apply least weight prune in the convolutional structure. Since there may be other layers between the convolutional layers that will change the output feature map numbers of the previous convolutional layer, we assume for two adjacent convolutional layers, they are independent. So that we can apply least weight prune for two dimensions of each convoutional layer.

3.1 Adaptive-Surgery

We create Adaptive-Surgery, a unified neural network compression solution. Adaptive-Surgery will take a compression ratio as an input, then compute the compression parameter

β for each layer. For each convolutional layer:

$$\beta = \frac{Targetdimensionlengthbeforepruning}{Targetdimensionlengthafterpruning}$$

For each fully-connected layer:

$$\beta = \frac{Neuronsnumberafterprune}{Neuronsnumberbeforeprune}$$

Since for convolution layer, we will apply least weight prune [14] on two dimensions, which means if we use a four dimensions tensor to mathematically represent this layer, the size of the tensor for each convolutional layer is $\frac{1}{\beta^2}$ of the original tensor. Meanwhile if we use a weight matrix to represent the inner product of two fully-connected layer, the size of the weight matrix after pruning is $\frac{1}{\beta^2}$ of the original weight matrix. Then we can have a comprehensive compression ratio for convolutional structure and fully-connected layer which is $1 - \beta^2$. So that for each input compression ratio:

$$\beta = \sqrt{1 - Compressionratio}$$

We present an example of compressing a neural network with three convolutional layer and three fully-connected layer. The detail is shown in Figure 3.1. The basic steps of compressing neural networks with Adaptive-Surgery can be summarized as below:

1. For the first convolutional layer that connected to the input data, Adaptive-Surgery will compute the summary of the the absolute value for each kernel and use this absolute value as the weights for each kernel, then prune the kernel with least weights based on the determined compressing ratio.

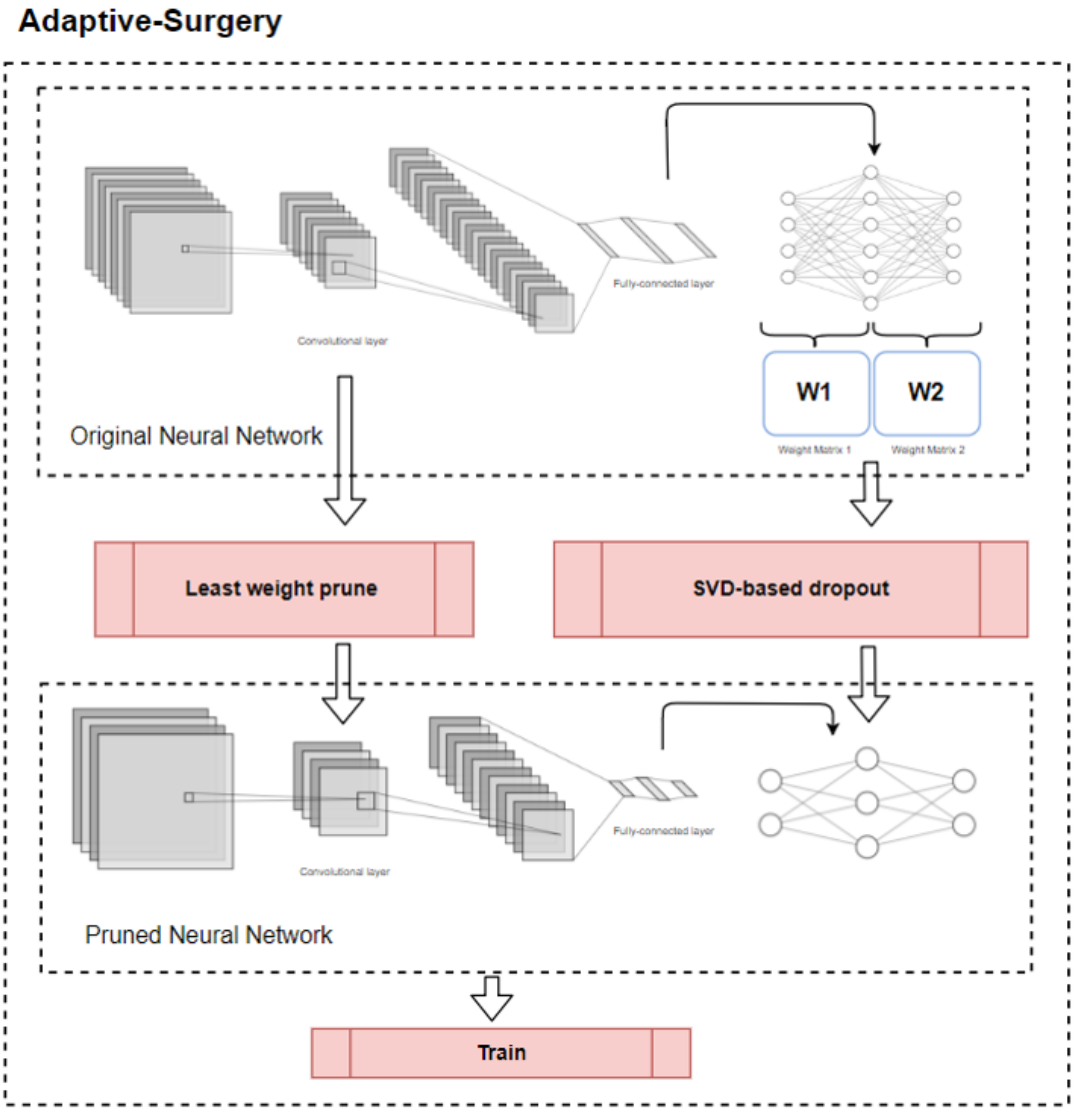


Figure 3.1: Example of how Adaptive-Surgery framework compress a DNN model. Blue boxes represent the weight matrices for fully-connected layer

2. For the second and following convolutional layer, since the pruning process will prune the output kernel for previous layer, which means we need to prune the current layer's kernel tensor in one dimension that corresponding to previous layer. Adaptive-Surgery will first use least weight prune to prune the dimension that corresponding to previous layer to the determined compression ratio, then prune the kernels that least weights in another dimension based on the β that compute from the input compression ratio.

3. For the first fully-connected layer that connected to the last convolutional layer:

   (a) If it is the only fully-connected layer, Adaptive-surgery will not compress it, since this layer is responsible for the generated result of this neural network.

   (b) If it is not the only fully-connected layer, but totally there are only two fully-connected layer, Adaptive-Surgery will target at the weight matrix between the adjacent fully-connected layer and using least weight prune to prune the dimension of the weight matrix corresponding to the first fully-connected layer, leave the dimension corresponding to the second fully-connected layer unaffected.

   (c) If there exist more than two fully-connected layers, Adaptive-Surgery will still using least weight prune to prune the dimension of the weight matrix corresponding to the first fully-connected layer. Then target at the adjacent weight matrices, two adjacent weight matrices are responsible for three fully-connected layers, so after Adaptive-Surgery applying SVD-based dropout on two adjacent weight matrices, the middle fully-connected layer will be pruned by β. Adaptive-Surgery will keep applying SVD-based dropout until it meet the final fully-connected layer.

3.2 SVD-based dropout

In this subsection, we will discuss the technical detail of SVD-based dropout. Adaptive-Surgery will compress each fully-connected layer in an iterative manner and enable this compression process to be applied on the whole fully-connected structure and using the property of Singular Value Decomposition to lower the loss of accuracy.

We want to discuss this compression method by giving an example of how SVD-based dropout compress a $3 \times 4 \times 3$ fully-connected structure. We donate the two adjacent weight matrices to be W_1 and W_2, and the bias matrices is b_1 and b_2. Then assume the input of this fully-connected structure is x and the output is y, so we have:

$$y = W_2 \cdot (W_1 \cdot x + b_1) + b_2$$

We assume $W_3 = W_2 \cdot W_1$, and the parameters in bias matrices are far less than weight matrices, then we have:

$$y \approx W_2 \cdot W_1 \cdot x = W_3 \cdot x$$

We apply Singular Value Decomposition on the matrix W_3, so that $W_3 = U_{m\times m} \cdot S_{m\times n} \cdot Vh_{n\times n}$, then we set $\beta = \frac{1}{2}$ and assume the neurons number for the middle fully-connected layer is k, then $U_{m\times m}$ and $S_{m\times n}$ will be pruned to be $U'_{m\times k/2}$ and $S'_{k/2\times n}$, we will have:

$$W'_2 = U'_{m\times k/2}$$

$$W'_1 = S'_{k/2\times n} \cdot Vh_{n\times n}$$

If in the pruning process, we keep as much parameters of the diagonal value of matrix $S_{m\times n}$, we can assume we keep as much as possible the information of original matrix, then we can

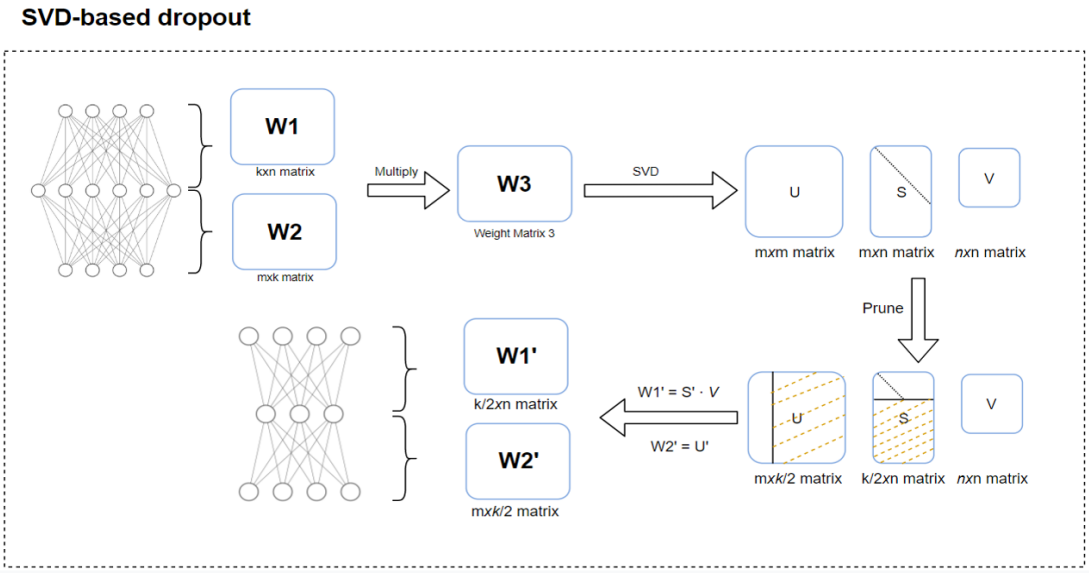


Figure 3.2: Process of SVD-based dropout compression. The black dotted line in the S matrix represent the diagonal none zero parameters. The orange dotted line in the U and S matrix represent the prune process.

assume $W_3 \approx W_3' = W_2' \cdot W_1'$. Once we donate the output of the pruned fully-connected structure to be y', we have:

$$y \approx y' = W_3' \cdot x = W_2' \cdot W_1' \cdot x$$

3.3 Least weight prune

In this subsection, we will discuss the technical detail of least weight prune and random prune. Adaptive-Surgery will compress each convolutional layer in an iterative manner and enable this compression process to be applied on the whole convolutional structure and using least weight prune to lower the loss of accuracy.

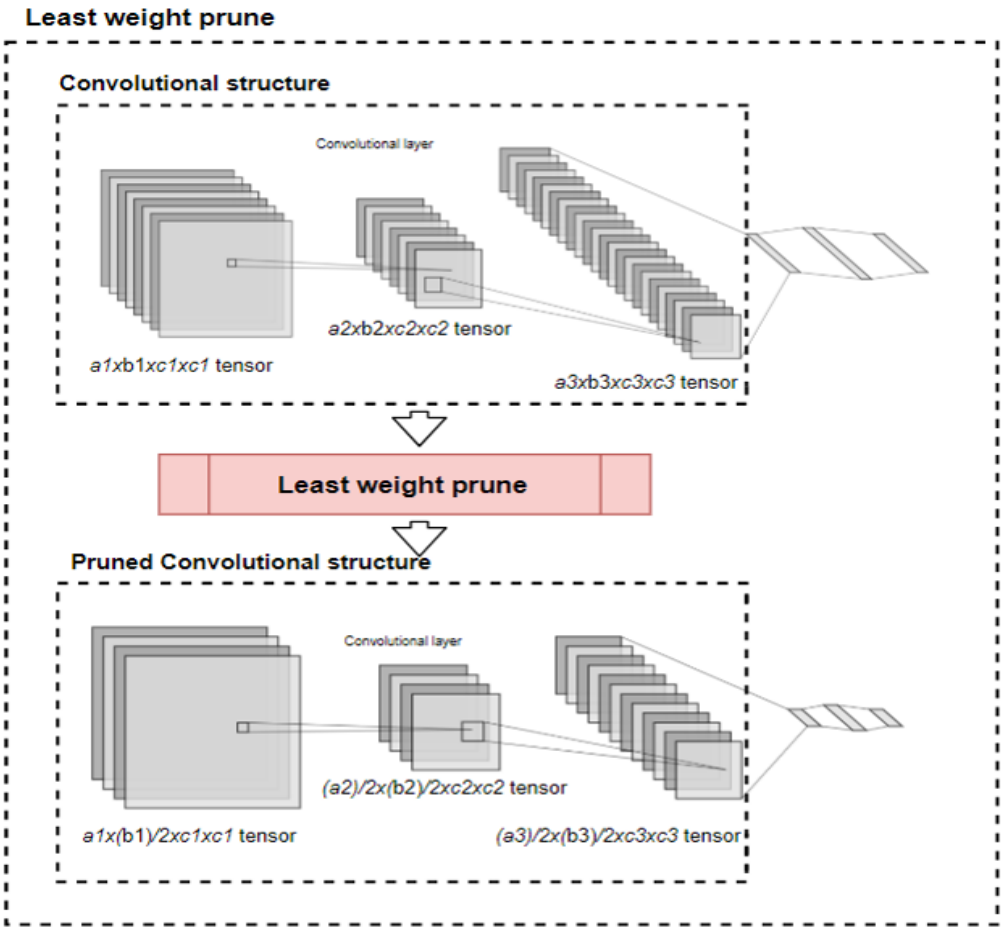


Figure 3.3: Overall process of least weight prune process for convolutional structure.

We want to discuss this compression method by giving an example of how least weight prune and random prune compress a convolutional structure with three convolutional layer. Since the GPU will process multiple neural network with same structure in parallel manner, we use the a dimension in the tensor to represent how many neural network are processing at the same time, which is also called batch size. For each convolutional layer, it will have several convolutional kernels, we use b dimension in the tensor to represent how many kernels in each layer. Meanwhile, we use c to represent the size of each kernel, kernel is also called feature map.

For the first convolutional layer, we want to keep the batch size for data input remain unaffected, so we only apply least weight prune on the b dimension. So for each kernel with size of c_1, we compute the summary of the absolute value for each element in the $c_1 \times c_1$ kernel and name this result to be the $weight$ for this kernel. Then Adaptive-Surgery will only prune the kernel will the least value of $weight$ repeatedly until the pruned structure reach the compression ratio.

For the second and following convolutional layers, we will apply least weight prune on both a and b dimensions until both dimensions reach the determined β.

Chapter 4

Evaluation

In the evaluation chapter, we first introduce the experiment setup by introducing the deep learning framework we use to develop and evaluate Adaptive-Surgery, and the device we used to test the execution time of different DNN models. We also give the details of the two DNN models we fed for Adaptive-Surgery as well as their dataset. Then we give the process of how we conducted the experiment followed by the experiment summary.

4.1 Experiment setup

4.1.1 Software

We use $Caffe$ [8] as deep-learning framework. Caffe is a deep learning framework made with expression, speed, and modularity in mind. It is developed by Berkeley AI Research (BAIR) and by community contributors. Yangqing Jia created the project during his PhD at UC Berkeley. Caffe is released under the BSD 2-Clause license.

4.1.2 Hardware

We use Raspberry Pi 3 Model B [5] for estimating the execution time performance of different Deep Neural Network models. The Raspberry Pi 3 Model B is the earliest model of the third-generation Raspberry Pi, it has following basic parameters:

- Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
- 1GB RAM

4.1.3 Alexnet

The original structure of Alexnet is designed in the paper named 'ImageNet Classification with Deep Convolutional Neural Networks' [1]. The first convolutional layer filters the $227 \times 227 \times 3$ input image with 96 kernels of size $11 \times 11 \times 3$ with a stride of 4 pixels (this is the distance between the receptive field centers of neighboring neurons in a kernel map). The second convolutional layer takes as input the (normalized and pooled) output of the first convolutional layer and filters it with 256 kernels of size $5 \times 5 \times 96$. The third, fourth, and fifth convolutional layers are connected to one another without any intervening pooling or normalization layers. The third convolutional layer has 384 kernels of size $3 \times 3 \times 256$ connected to the (normalized, pooled) outputs of the second convolutional layer. The fourth convolutional layer has 384 kernels of size $3 \times 3 \times 384$, and the fifth convolutional layer has 256 kernels of size $3 \times 3 \times 384$. The fully-connected layers have 4096 neurons each.

We conduct experiments on a modified Alexnet model [12] that implemented by Adil Moujahid, he uses a DNN model which is similar to the original Alexnet, but change

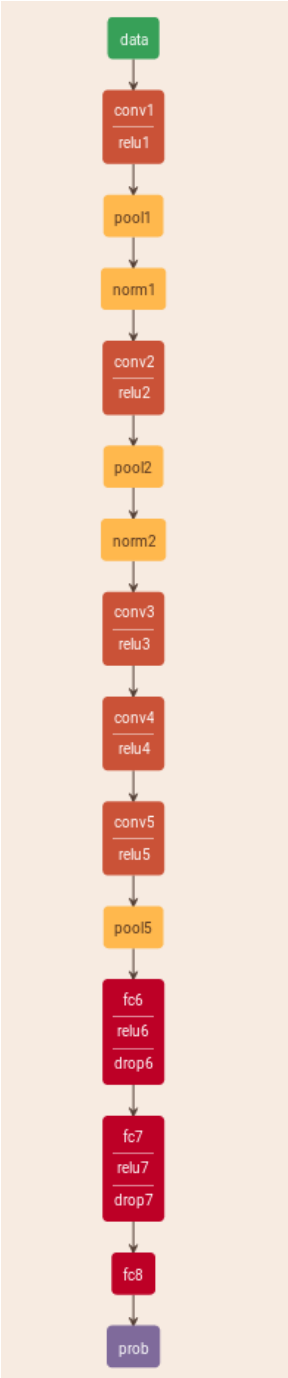


Figure 4.1: Modified Alexnet architecture

the order of the pool layer and Relu layer for each convolutional layer, while the last fully-connected layer is modified to have only 2 neurons, since this model is designed to classifying between images of cats and dogs. The dataset is download from Kaggle's Cats and Dogs challenge [17].

4.1.4 CIFAR-10

CIFAR-10 is a dataset [13] that consists of 60000 32×32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. We use a model called $cifar10-quick$ that built in Caffe framework to classify the CIFAR-10 dataset.

The $cifar10-quick$ consist of three convolutional layers, three pool layers, three ReLU layers and two fully-connected layers. The first convolutional layer filters the $32 \times 32 \times 3$ input image with 32 kernels of size $5 \times 5 \times 3$ with a stride of 2 pixels. The second convolutional layer takes as input the (rectified and pooled) output of the first convolutional layer and filters it with 32 kernels of size $5 \times 5 \times 32$. The third convolutional layer takes as input the (rectified and pooled) output of the second convolutional layer and filters it with 64 kernels of size $5 \times 5 \times 32$. All the three convolutional layers are followed by the Max-pooling layer and ReLU layer, the property of these two kinds of layer is defined in the Caffe framework, since the Adaptive-Surgery only focus on convolutional structure, fully-connected stucture and their combination, we will not spend much time on dicussing the Max-pooling layer and ReLU layer. Followed by the third convolutional layer is the first fully-connected layer, it has 64 neurons. The second fully-connected layer has 10 neurons

Figure 4.2: The structure of cifar10-quick

to classify the ten classes for the input image dataset. The dataset is download from the Cifar10 challenge [18].

4.2 Architecture

Given the original neural network structure and parameters as well as the input compression ratio, Adaptive-Surgery can compute β for each layer and automatically compress the original neural network by β, the generated neural networks can be directly implemented on the computer or embedded system. In the experiments, we first set three compression ratio for Adaptive-Surgery to compress modified Alexnet and Cifar10-quick. After the Adaptive-Surgery compress the original pre-trained DNN models to three compression ratio, we measure the model size and the accuracy performance on the generated models. Then we test the execution time for the generated models on the Raspberry Pi 3. For three different compression ratio, we also build three new DNN models, whose initial parameters are randomly chosen, and we train these models using the same training police on the same dataset as the model that generated by Adaptive-Surgery does. And compare the accuracy between the models that generated by Adaptive-Surgery and the accuracy that trained from the models whose initial parameters are randomly chose.

4.3 Experiment result

We test the execution time for each structure in the two DNN models. We can tell from the figures that the convolutional structure as well as fully-connected structure are

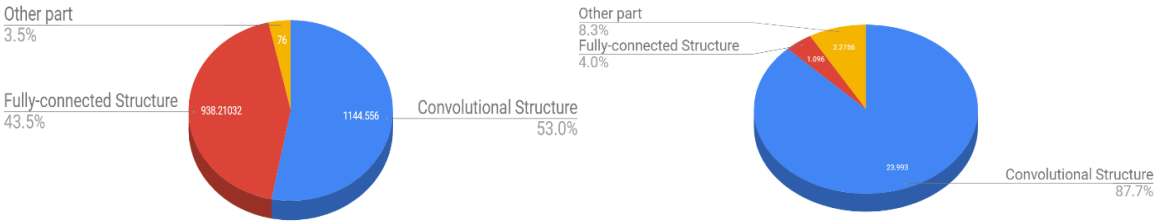


Figure 4.3: The pie charts of execution time for each structure in the modified Alexnet (Left) and Cifar10-quick (Right)

Table 4.1: Performance summary for Alexnet

Model	Size (MB)	Actual Compression Ratio (%)	Execution time (ms)	Speedup	Accuracy
Alexnet-modified	227.5	0%	2158.85	1×	90.64%
Alexnet-modified (43.75%)	128	43.7%	1252.46	1.72×	92.44%
Alexnet-modified (75%)	56.5	75.2%	613.974	3.52×	90.02%
Alexnet-modified (93.75%)	14.3	93.71%	128.902	16.75×	55.24%

responsible for over 90% of the execution time in the original DNN models. This means if we apply compression methods only on convolutional structure and fully-connected structure, the whole DNN model can still research determined compression ratio.

We use Adaptive-Surgery to compress Alexnet to three degree, which are 'Rare', 'Medium' and 'Well-Done', and the corresponding compression ratio are 56.25%, 25% and 6.25%. And the below two table are the model size, actual compression ratio, excution time, speedup and accuracy for the DNN models that generated by Adaptive-Surgery.

So we can tell from the first two tables that:

Table 4.2: Performance summary for Cifar10-quick

Model	Size(KB)	Actual Compression Ratio (%)	Execution time (ms)	Speedup	Accuracy
Cifar10-quick	583.3	0%	27.3646	1×	71.56%
Cifar10-quick (43.75%)	331	43.3%	18.1045	1.51×	73.60%
Cifar10-quick (75%)	149.9	74.3%	10.384	2.64×	70.17%
Cifar10-quick (93.75%)	39.9	93.16%	4.887	5.60×	64.35%

1. For three user input compression ratios, we can compute the actual compression ratio from the actual pruned model size. The experiment result for actual compression ratios shows that Adaptive-Surgery can automatically compress the original DNN models to meet the determined compression ratios.

2. Since the execution time should be proportional to the amount of remaining parameters after prune, which means for three compression ratios: 43.75%, 75% and 93.75% the ratio of remaining amount of parameters to the original models' parameters should be 56.25%, 25% and 6.25%. Then the corresponding speed up should be $1/56.25\% \approx 1.78$, $1/25\% = 4$ and $1/6.25\% = 16$. And the experiment results show that the pruned modified Alexnet generated by Adaptive-Surgery can achieve determined speedup. But for pruned Cifar10-quick, when the ratio of the other structure rather than convolutional structure and fully-connected structure can not by ignored, it will influence the actual speedup for pruned models, which explains why speedup of Cifar10-quick is lower than the determined speedup.

3. For the accuracy performance, when both models reach the 43.75% compression ratio, the accuracy of the pruned models are better than that of original models. This is mainly because we use least weight prune and SVD-based dropout prune the less important parameters of the original models, and train the pruned models which have all the significant survival parameters of the original DNN models, this solves the overfitting problems of the original DNN models by pruning the redundant parameters.

Table 4.3: Adaptive-Surgery model Accuracy (%) and Initial random value model Accuracy (%) for modified Alexnet

Model	Compression Ratio = 0%	Compression Ratio = 43.75%	Compression Ratio = 75%	Compression Ratio = 93.75%
Adaptive-Surgery model	90.64%	92.44%	90.02%	55.24%
Initial random value model	90.64%	87.95%	82.65%	57.33%

Table 4.4: Adaptive-Surgery model Accuracy (%) and Initial random value model Accuracy (%) for Cifar10-quick

Model	Compression Ratio = 0%	Compression Ratio = 43.75%	Compression Ratio = 75%	Compression Ratio = 93.75%
Adaptive-Surgery model	71.56%	73.60%	70.17%	64.35%
Initial random value model	71.56%	70.64%	68.51%	61.62%

Then we evaluate the accuracy performance of the models that generated by Adaptive-Surgery and of that whose initial parameters are randomly chosen while having exactly same structure as the pruned DNN models.

Then we can tell from the last two tables that: For most of the situations, the trained models that generated in Adaptive-Surgery have better accuracy performance than trained model whose initial parameters for DNN models are randomly chosen. This means that applying least weight prune for convolutional structure and SVD-based dropout for fully-connected structure can lower the accuracy penalty for compression a DNN model. The accuracy of modified Alexnet with compression ratio to be 93.75% is messed up, mainly because in the compressing process of Adaptive-Surgery, there are too many parameters are pruned to guarantee the accuracy.

Chapter 5

Conclusions

Prior works applied singular value decomposition and dropout compression methods for fully-connected structure, or pruned the feature map by magnitude of kernel weights for convolutional structure. They only focused on one specific DNN structure. Other than that, prior framework needed numerous time in the compressing process to ensure the least accuracy penalty and they could not take a compression ratio as an input. However, this thesis proposes an idea about combining the different compression methods to generate a unified approach that compresses the whole DNN model. Moreover, we successfully combine singular value decomposition and dropout to be a new method called SVD-based dropout which turns out to be a efficient method for compressing fully-connected structure. Plus, we create a framework called Adaptive-Surgery that can take user input compression ratio and automatically decide the compression parameter β for each convolutional layer and fully-connected layer and compresses them based on β. The compressed models that generated by Adaptive-Surgery can be directly implemented on the Raspberry Pi 3 Model B, which

has constrained processing resource. The experiment results show that target two DNN models (Alexnet and Cifar10-quick) can be compressed by Adaptive Surgery to three input compression ratios (43.75%, 75% and 93.75%) with soft accuracy penalty. The experiment results also prove that compared with the models whose initial parameters are randomly chosen, those models that generated by Adaptive-Surgery have less accuracy penalty. More importantly, we can manually set the compression ratio for Adaptive-Surgery, which means in order to implement DNN models on resource constrained devices with specific requirements for real-time correctness, Adaptive-Surgery can compress the DNN models to meet the requirements.

For the future work, Adaptive-Surgery could be evaluated on more than two DNN models to prove its performance. Although our SVD-based dropout have a great performance on compressing the fully-connected structure, it can be improved by having a better way to represent the bias matrices in the pruned model rather than ignore the influence of bias matrices. Also, if we can combine two different compression methods to generate a unified compression framework, researchers can try more combinations of different compression methods for compressing the DNN to achieve a unified approach. Last but not the least, in the evaluation of this thesis, we do the experiments to test for three compression ratios. However, since out framework can support arbitrary ratios, it will be interesting to try with more diverse set of compression ratios and analyze their results.

Bibliography

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton, *ImageNet Classification with Deep Convolutional Neural Networks*, NIPS 2012

[2] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao and Karol Zieba, *End to End Learning for Self-Driving Cars*, arXiv:1604.07316, 2016.

[3] Sourav Bhattacharya and Nicholas D. Lane, *Sparsification and Separation of Deep Learning Layers for Constrained Resource Inference on Wearables*, in proceedings of 14th ACM Conference on Embedded Network Sensor Systems CD-ROM. Pages 176-189

[4] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever and Ruslan Salakhutdinov, *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*, JMLR 2014.

[5] Raspberry Pi 3 Model B,
`https://www.raspberrypi.org/products/raspberry-pi-3-model-b/`

[6] Deep Neutral Networks,
`https://www.techopedia.com/definition/32902/deep-neural-network`.

[7] Shuochao Yao, Yiran Zhao, Aston Zhang, Lu Su and Tarek Abdelzaher, *DeepIoT: Compressing Deep Neural Network Structures for Sensing Systems with a Compressor-Critic Framework*, arXiv:1706.01215, 2017.

[8] Caffe,
`http://caffe.berkeleyvision.org/`.

[9] Kaipeng Zhang, Zhanpeng Zhang, Zhifeng Li, and Yu Qiao, *Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks*. arXiv:1604.02878, 2016.

[10] Akhil Mathur , Nicholas D. Lane, et al. *DeepEye: Resource Efficient Local Execution of Multiple Deep Vision Models using Wearable Commodity Hardware*, in proceeding of the 15th Annual International Conference on Mobile Systems, Applications, and Services. 2017.

[11] Nicholas D. Lane, Petko Georgiev, Lorena Qendro, *DeepEar: Robust Smartphone Audio Sensing in Unconstrained Acoustic Environments using Deep Learning*, in proceedings of the ACM International Joint Conference on Pervasive and Ubiquitous Computing. 2015.

[12] A Practical Introduction to Deep Learning with Caffe and Python, `http://adilmoujahid.com/posts/2016/06/introduction-deep-learning-python-caffe/`.

[13] The CIFAR-10 dataset, `https://www.cs.toronto.edu/ kriz/cifar.html`.

[14] Hao Li, Asim Kadav et al. *Pruning Filters For Efficient ConvNets*, ICLR, 2017.

[15] G. H. GOLUB and C. REINSCH. *Singular Value Decomposition and Least Squares Slutions.*, Linear Algebra, 1971.

[16] Understanding Dimension Reduction with Principal Component Analysis (PCA), `https://blog.paperspace.com/dimension-reduction-with-principal-component-analysis/`.

[17] Dogs vs. Cats dataset. `https://www.kaggle.com/c/dogs-vs-cats/data`

[18] Cifar10 dataset. `http://www.cs.toronto.edu/ kriz/cifar.html`