DNN WEIGHT COMPRESSION METHOD WITH ADMM FRAMEWORK

A Thesis

by RACHEL HSINGTZE YEH

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair Name,	Chao Tian
Committee Member,	Jean-Francois Chamberland-Tremblay
	I-Hong Hou
	Zhangyang Wang
Head of Department,	Miroslav M. Begovic

December 2019

Major Subject: Electrical Engineering

ABSTRACT

Deep neural networks (DNNs) is a powerful technique evolved to the state-of-the-art technique for computer vision tasks. The "deep compression" is introduced to overcome the significant problem in memory- and computational-efficient. The basic pipeline of compression is the three-stage model including pruning/fine-tuning, quantization/clustering, and encoding, and the Alternating Direction Method of Multipliers (ADMM) algorithm has been applied into pruning stage to improve the performance. Furthermore, the quantization/clustering is used with a 2-bit representation jointly to reduce the storage. Lastly we auto-adjust the hype-parameters, apply classification on the gradients and filter the whole layers to increase the weight reduction ratio and solving the time-consuming problems. The algorithm is training on the LeNet-5 model using the MNIST dataset and has 222.3x weight number reduction as the result.

ACKNOWLEDGMENTS

I would like to thank my committee chair, Dr. C. Tian, and my committee membes, Dr. J.F. Chamberland, Dr. Z. Wang, and Dr. I. Hou, for their guidance and support throughout the course of this research.

Thanks also go to my friends and colleagues and the department faculty and staff for making my time at Texas A&M University a great experience.

Finally, thanks to my mother and father for their encouragement and my siblings for their patience and love.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supervised by a thesis committee consisting of Professors Tian, Chao and Professor Chamberland, Jean-Francois and Professor Hou, I-Hong of the Department of ECEN, and Professor Wang, Zhangyang of the Department of CSCE.

The data analyzed for Chapter 5 was provided by Professor Tian, Chao. The analyses depicted in Chapter 3 and 4 was reproduced in part from Shaokai Ye and Tianyun Zhang published in 2018 in an article listed in the Bibliography.

All other work conducted for the thesis was completed by the student independently.

Funding Sources

There are no outside funding contributions to acknowledge related to the research and compilation of this document.

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION

Within the past decade, DNNs have been developed widely and applied to various fields such as image classification, speech recognition, game playing, and applications. Back to 1990's, Convolutional Neural Networks (CNNs) was first introduced by LeCun *et al.* called LeNet-5 model which has less than 1M parameters to classify the handwritten digits [1], later in 2012 the ImageNet is standing out with 60M parameters by Krizhevsky *et al.* [2], and today we have AlexNet Caffe-model with over 200MB and VGG-16 Caffe-model with over 500MB. The large-scale/dimension neural networks are considered and used in deep learning algorithms, which brings the attention of how critical and difficult for their parameter storage and computational cost in real-time applications, e.g. online learning or incremental learning [3]. To mitigate these limitations, the DNN compression model and techniques have been widely investigated.

The first compression method, network pruning, was introduced by LeCun in Optimal Brain Damage when developing the CNN in 1990 [4]. This idea is based on finding the redundant weights among large parameters in the network and get rid of them. The early trend of pruning in 1990's is using second-order Taylor expansion to select redundant parameters and remove the individual parameters [5] [4] or entire units [6]. Unfortunately, this method becomes impractical if dealing with large-scale network because it requires the partial- (or complete-) Hessian matrix and brings the expensive cost of computational in fine-tuning [7] [8]. Therefore, plenty of different methods is developed to improve the pruning method, e.g. parameter sharing and quantization can robust various setting, low-rank factorization and compact convolutional filters can be implemented in CPU/GPU easily, and knowledge distillation can train a compact neural network with distilled knowledge [3].

2. BACKGROUND OF WEIGHT PRUNING, CLUSTERING, AND QUANTIZATION

Parameter pruning and sharing are well-investigated in prior works with different algorithms such as vector quantization, clustering, binary coding, and sparse constraints [3]. Han *et al.* [9] [10] provide a pioneering work on iterative weight pruning pipeline as Fig. 2.1 which is reprinted from [3]. The compression method including three parts: the first stage is training, pruning and fine-tuning; the second stage is clustering, vector quantization, and retraining; and the last stage is binary encoding. The weight pruning method in the first stage is developed widely. For example, [11] uses hash bucket to share the parameter value and creates HashNet model; [12] prunes entire convolutional filters; [8] finds network cost function and uses oracle pruning method; [9] uses scalar quantization and centroid fine-tuning to reach the goal.

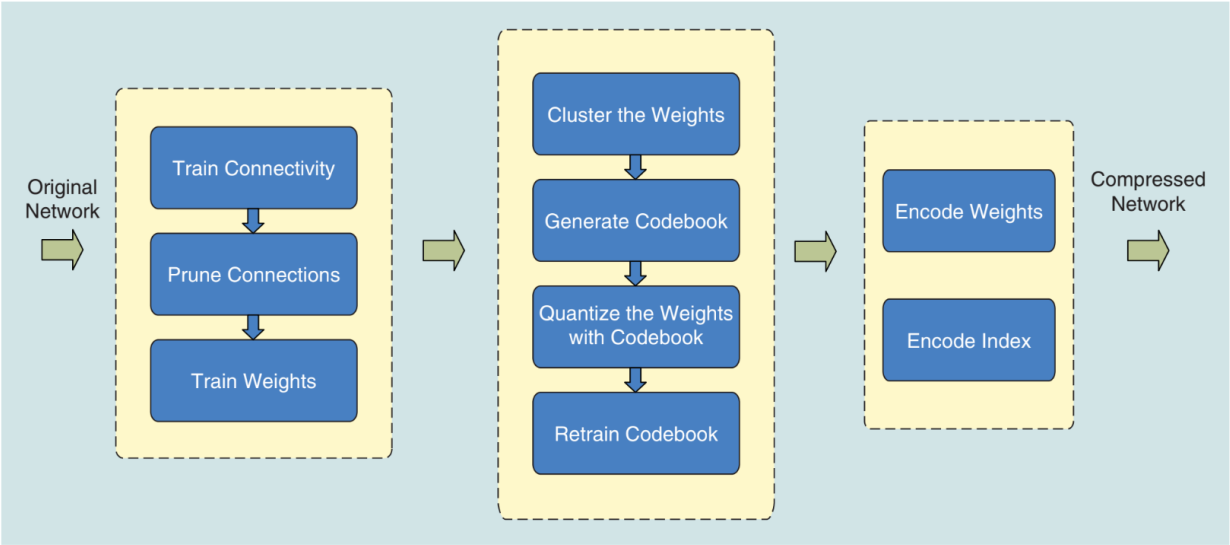


Figure 2.1: The Three-Stage Compression Method: Pruning, Quantization/Clusteriung, and Encoding. This figure is reprinted from [3].

Putting weight pruning/quantization and fine-tuning in the parallel process can eliminate premature pruning errors while taking both advantages of pruning and quantization, such as [13] has

CLIP-Q method which performing the weight pruning, quantization and fine-tuning in the parallel process. Furthermore, using different loss functions can get better computational results, for example, [14] applies the ADMM algorithm in pruning loss function to get better computation results. Therefore, [15] creates a unified framework which combines the ADMM with quantization and clustering and successfully has 167x weight reduction and 1910x storage reduction in LeNet-5 without any significant accuracy loss.

The procedure of compression is showing in Fig.2.2. First, the redundant weights will be pruned as zero weights in (a) and will not be quantized and clustered anymore in future steps. Moving to next, the quantization level set in Fig.2.2 is calculated as $Q = \{-1, -0.5, 05, 1\}$ with equal steps of 0.5 and each non-zero weights will be quantized to the closest level (c), then based on 2-bit representation the value will be stored in hardware as (d). For clustering, the centroid values (f) will be calculated and the non-zero weights will be clustered (e). The quantization and clustering are similar to each other, but the centroid values are more flexible in clustering [15].

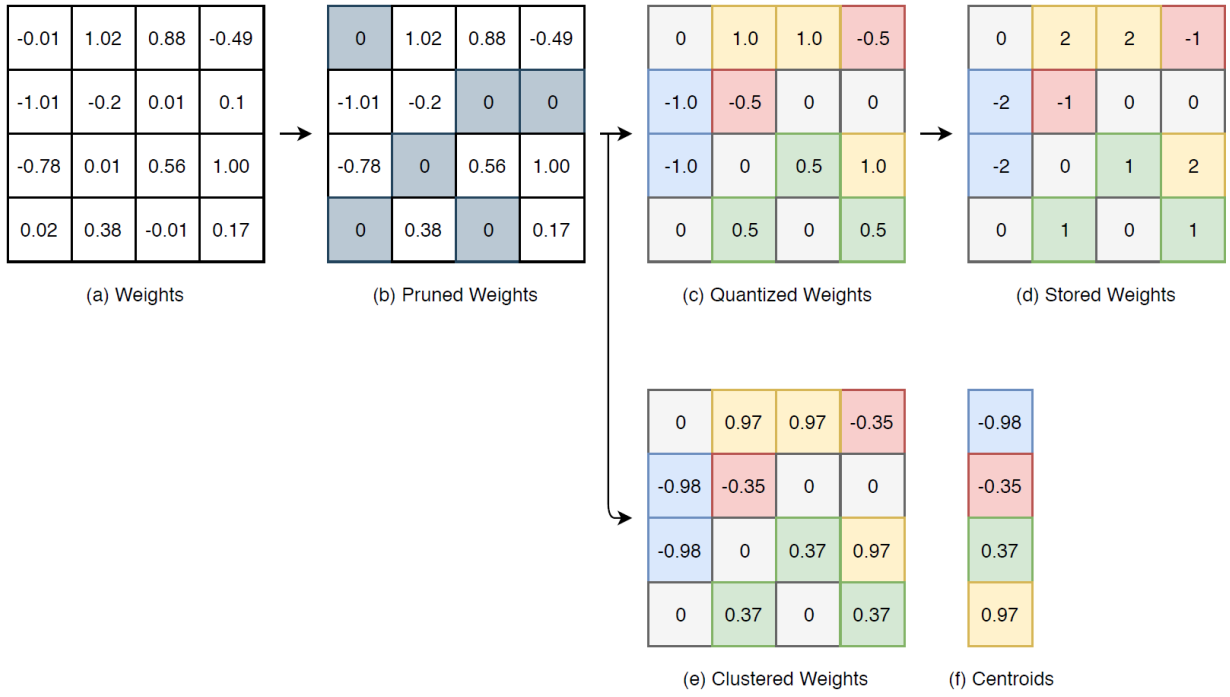


Figure 2.2: Procedure of Compression: (a-b) weights pruning, (c-d) weight quantization, (e-f) weight clustering.

3. BACKGROUND OF ADMM FRAMEWORK

ADMM (Alternating Direction Method of Multipliers) is an algorithm proposed by Gabay *et al.* [14] that solves convex optimization problems by decomposing them into several sub-problems. Recently it has been demonstrated and used in non-convex problems powerfully and widely for many applications. Consider an optimization problem in generic form

$$\begin{aligned} \min_{x} \quad & f(x) + g(x) \\ \text{subject to} \quad & x \in C \end{aligned} \tag{3.1}$$

If $f(x)$ is differentiable and take $g(z)$ as indicator function of constraint C, then the optimization problem in ADMM form can be rewritten as

$$\begin{aligned} \min_{x,z} \quad & f(x) + g(z) \\ \text{subject to} \quad & x - z = 0 \end{aligned} \tag{3.2}$$

Thus the problem is decomposed into two sub-problems reflect on x and z by applying augmented Lagrangian: $\min_x f(x) + q_1(x)$ and $min_z g(z) + q_2(z)$ where both $q_1(x)$ and $q_2(z)$ are quadratic function of its argument [16]. First sub-problem can be solved regularly as minimizing $f(x)$ using stochastic gradient descent if the function f and q_1 are differentiable; and the optimization problem can be solved analytically by exploring the properties of g if g has special structure, e.g. g is a regularizer [14].

3.1 ADMM Based Pruning

Consider a set of training examples with input x and output label y: $D = \{X = \{x_0, x_1, ..., x_N\}$, $Y = \{y_0, y_1, ..., y_N\}\}$, the network's parameter is given as $W = \{(w_1, b_1), ..., (w_i, b_i)\}$ where w_i is the collection of weights in i^{th} layer and b_i is the collection of biases in i^{th} layers [8], then the

lost function can be represented as $f(\{W_i\}_{i=1}^N, \{b_i\}_{i=1}^N)$.

The goal of pruning method is removing the redundant weights while the accuracy loss is negligible, therefore we minimize the loss function of network subject to cardinality constraints of weights in each layer and having the following optimization problems where $card(\cdot)$ represents the non-zero elements in matrix argument and α_i gives desired number of weights in i^{th} layer [14]:

$$\begin{aligned} \min_{\{W_i\},\{b_i\}} \quad & f(\{W_i\}_{i=1}^N, \{b_i\}_{i=1}^N) \\ \text{subject to} \quad & card(W_i) \leq \alpha_i \quad i = 1, ..., N \end{aligned} \tag{3.3}$$

The constraint in Eq.(3.3) can be written as $W_i \in S_i$ where $S_i = W_i | card(W_i) \leq \alpha_i$ represents the non-zero elements that less than or equal to specific number of weights. Since we have non-convex constraint with difficulty to solve, adding an indicator function $g(\cdot)$ of S_i is necessary in problem. Let g_i given as $g_i(W_i) = 0$ if $card(W_i) \leq \alpha_i$ otherwise $g_i(W_i) = +\infty$, then the problem can be reformatted without constraints as $\min_{\{W_i,b_i\}} f(\{W_i\}, \{b_i\}) + \sum_{i=1}^N g_i(W_i)$. Notice that the term with indicator function is non-differentable, based on ADMM optimization form via decomposition, the problem is reformatted again equivalently as

$$\begin{aligned} \min_{\{W_i\},\{b_i\}} \quad & f(\{W_i\}, \{b_i\}) + \sum_{i=1}^N g_i(Z_i) \\ \text{subject to} \quad & W_i = Z_i \quad i = 1, ..., N \end{aligned} \tag{3.4}$$

Then the problem's augmented Lagrangian is clearly formatted as method of multipliers with Lagrange multiplier Λ_i, penalty parameters $\{\rho_1, ..., \rho_N\}$, Frobenius norm $\|\cdot\|_F^2$ and scaled dual variable $U_i = \frac{1}{\rho_i}\Lambda_i$ as:

$$\begin{aligned} L_\rho(\{W_i\}, \{b_i\}, \{Z_i\}, \{\Lambda_i\}) = f(\{W_i\}, \{b_i\}) + \sum_{i=1}^N g_i(Z_i) + \sum_{i=1}^N tr[\Lambda_i^T(W_i - Z_i + U_i)] \\ + \sum_{i=1}^N \frac{\rho_i}{2}\|U_i\|_F^2 \end{aligned} \tag{3.5}$$

The algorithm consists the iterations of two minimization and one dual update as Eq.(3.6) for $k = 0, 1, ...$ where dual variable update fellows the same step size as augmented Lagrangian parameter ρ. The processes repeat until $\|W_i^{k+1} - Z_i^{k+1}\|_F^2 \leq \epsilon_i$ and $\|Z_i^{k+1} - Z_i^k\|_F^2 \leq \epsilon_i$ are reached.

$$\begin{aligned} \{W_i^{k+1}, b_i^{k+1}\} &:= \underset{\{W_i\},\{b_i\}}{\operatorname{argmin}} L_\rho(\{W_i\}, \{b_i\}, \{Z_i^k\}, \{U_i^k\}) \\ \{Z_i^{k+1}\} &:= \underset{\{Z_i\}}{\operatorname{argmin}} L_\rho(\{W_i^{(k+1)}\}, \{b_i^{(k+1)}\}, \{Z_i\}, \{U_i^k\}) \\ U_i^{k+1} &:= U_i^k + W_i^{(k+1)} - Z_i^{(k+1)} \end{aligned} \tag{3.6}$$

ADMM algorithm decomposes the original optimization problem Eq.(3.3) into two sub-problems as Eq(3.7) and Eq(3.8) which are transformed from two minimization in Eq.(3.6). In sub-problem Eq.(3.7), the first $f(\cdot)$ term is the regular loss function and the second term is represented as L_2 regularizer, both of them are differentiable and can be solved using stochastic gradient descent. In sub-problem Eq.(3.8), $g_i(\cdot)$ is the indicator function of non-convex set S_i and brings the solution that $Z_i^{k+1} = \Pi_{S_i}(W_i^{k+1} + U_i^k)$ where $\Pi_{S_i}(\cdot)$ is the Euclidean projection of S_i, this projection keeps the maximum magnitude of α_i numbers of elements within $W_i^k + 1 + U_i^k$ and prune the rest of elements (as zero). [16] [14]

$$\min_{\{W_i\},\{b_i\}} \quad f(\{W_i\}, \{b_i\}) + \sum_{i=1}^{N} \frac{\rho_i}{2} \|W_i - Z_i^k + U_i^k\|_F^2 \tag{3.7}$$

$$\min_{\{Z_i\}} \quad \sum_{i=1}^{N} g_i(Z_i) + \sum_{i=1}^{N} \frac{\rho_i}{2} \|W_i^{k+1} - Z_i + U_i^k\|_F^2 \tag{3.8}$$

3.2 ADMM Based Pruning with Clustering/Quantization

Consider the pruning process with clustering and quantization, the constraint in the above optimization problem needs to adjust and re-define. The weight clustering gives us a constraint that remaining weights W_i should contain less than $M_i = 2^n$ different values, where n represents the number of bits. And the quantization gives another constraint that remaining weights

are not only limited by M_i different values but also only taken from desired quantization level set: $Q = \{Q_1, Q_2, ..., Q_{M_i}\}$, where Q values are equal distance. Therefore combine the weight pruning problem with clustering and quantization, the overall optimization problem can be defined as equation below (S_i is the number of non-zero parameters less than α_i and S_i' is the non-zero weights only take values from set Q) [15].

$$\begin{aligned} \min_{\{W_i\},\{b_i\}} \quad & f(W_{i i=1}^{N}, b_{i i=1}^{N}) \\ \text{subject to} \quad & W_i \in S_i, \quad W_i \in S_i', \quad i = 1, ..., N \end{aligned} \tag{3.9}$$

Transferring the problem to ADMM form, two indicator functions $g_i(\cdot)$ and $h_i(\cdot)$ have been defined reflecting with S_i and S_i' such that $g_i(W_i) = 0$ if $W_i \in S_i$ and $h_i(W_i) = 0$ if $W_i \in S_i'$ otherwise $g_i(W_i) = h_i(W_i) = +\infty$, and the ADMM form is given by

$$\begin{aligned} \min_{\{W_i\},\{b_i\}} \quad & f(\{W_i\}, \{b_i\} + \sum_{i=1}^{N} g_i(Z_i) + \sum_{i=1}^{N} h_i(Y_i) \\ \text{subject to} \quad & W_i = Z_i, W_i = Y_i, i = 1, .., N \end{aligned} \tag{3.10}$$

The augmented Lagrangian of the problem can decompose it into three sub-problems with scaled dual variables U_i^k and V_i^k as shown below. For each iteration, dual variables are updating as $U_i^{k+1} := U_i^k + W_i^{k+1} - Z_i^{k+1}$ and $V_i^{k+1} := V_i^k + W_i^{k+1} - Y_i^{k+1}$ and repeating until satisfy $\|W_i^{k+1} - Z_i^{k+1}\|_F^2 \leq \epsilon_i$, $\|Z_i^{k+1} - Z_i^k\|_F^2 \leq \epsilon_i$, $\|W_i^{k+1} - Y_i^{k+1}\|_F^2 \leq \epsilon_i$ and $\|Y_i^{k+1} - Y_i^k\|_F^2 \leq \epsilon_i$:

$$\min_{\{W_i\},\{b_i\}} \quad f(W_{i i=1}^{N}, b_{i i=1}^{N}) + \frac{\rho_i}{2}\|W_i - Z_i^k + U_i^k\|_F^2 + \sum_{i=1}^{N} \frac{\rho_i}{2}\|W_i - Y_i^k + V_i^k\|_F^2 \tag{3.11}$$

$$\min_{\{Z_i\}} \quad \sum_{i=1}^{N} g_i(Z_i) + \sum_{i=1}^{N} \frac{\rho_i}{2}\|W_i^{k+1} - Z_i + U_i^k\|_F^2 \tag{3.12}$$

$$\min_{\{Z_i\}} \quad \sum_{i=1}^{N} h_i(Y_i) + \sum_{i=1}^{N} \frac{\rho_i}{2}\|W_i^{k+1} - Y_i + V_i^k\|_F^2 \tag{3.13}$$

It's clear the first sub-problem can be solved by stochastic gradient descent since both terms are

convex and differentiable. The second and third problem can be solved from indicator functions as $Z_i^{k+1} = \Pi_{S_i}(W_i^{k+1} + U_i^k)$ and $Y_i^{k+1} = \Pi_{S_i'}(W_i^{k+1} + V_i^k)$.

4. BACKGROUND OF ADMM ALGORITHMS IN DNN

The algorithm is applying to the LeNet-5 model with the MNIST dataset and AlexNet with the ImageNet dataset. The iterative updates during three sub-problem solving are high computational complexity, thus we separate the process into two parts: weight pruning and weight clustering/quantization [15].

4.1 Weight Pruning

For weight pruning, the first step is exploring the initial α_i value in each layer. Then the data is training according to update $\{W_i\}$ and $\{b_i\}$ by solving the loss function in each k iterations as $\left[\min_{\{W_i\},\{b_i\}} f(\{W_i\}_{i=1}^N, \{b_i\}_{i=1}^N) + \sum_{i=1}^N \frac{\rho_i}{2}\|W_i - Z_i^k + U_i^k\|_F^2\right]$. Within each iteration training, the Euclid mapping Z_i^{k+1} and U_i^{k+1} can be calculated and updated using the defined equation in the previous section.

Algorithm 1 Weight Pruning Method

```
Input: MNIST dataset
Initialization: Determine α_i for each layer
for k in ADMM iterations do
    Update {W_i} and {b_i} by solving:
        min_{{W_i},{b_i}} f({W_i}_{i=1}^N, {b_i}_{i=1}^N) + Σ_{i=1}^N (ρ_i/2)‖W_i − Z_i^k + U_i^k‖_F^2
    Save gradients as vector to classify
    for i in each layers do
        Update Z_i:   Z_i^{k+1} = Π_{S_i'}(W_i^{k+1} + U_i^k)
        Update V_i:   U_i^{k+1} = U_i^k + W_i^{k+1} − Z_i^{k+1})
    end for
end for
Retrain rest of the weights
Label redundant and remaining weights and apply classifiers on pull-out gradients
```

4.2 Weight Quantization

After pruning the redundant weights (zero weights), rest of the non-zero weights can be clustered and quantized by n bits, i.e. 2^n different weights. Starting with initialization, M_i and q_i of quantization levels are exploring in each layer. The set of quantization levels is given as $\{-\frac{M_i}{2}q_i, ..., -2q_i, -q_i, q_i, 2q_i, ..., \frac{M_i}{2}q_i\}$ with equal distance. The quantization step is given as a square error between weight and distance to its closet level as $\sum_j |w_i^j - f(w_i^j)|^2$. Then for each k in ADMM iterations, solving $\left[\min_{\{W_i\},\{b_i\}} f(\{W_i\}_{i=1}^N, \{b_i\}_{i=1}^N) + \sum_{i=1}^N \frac{\rho_i}{2}\|W_i - Y_i^k + V_i^k\|_F^2\right]$ as loss function by updating $\{W_i\}$ and $\{b_i\}$; and within each iteration training, the Euclid mapping Y_i^{k+1} and V_i^{k+1} can also be updated by the defined equation in the previous section. After exploring the set of quantization level, quantize $\alpha\%$ of weight to its closest level using regular training process and retrain the remaining weights.

Algorithm 2 Weight Pruning Method with Clustering/Quantization

Input: Data after Weight Pruning
Initialization: Determine M_i number of cluster in each layer
    find q_i as quantization level set in each layer $\{-\frac{M_i}{2}q_i, ..., -2q_i, -q_i, q_i, 2q_i, ..., \frac{M_i}{2}q_i\}$
for k in ADMM iterations **do**
    Update $\{W_i\}$ and $\{b_i\}$ by solving:
        $\min_{\{W_i\},\{b_i\}} f(\{W_i\}_{i=1}^N, \{b_i\}_{i=1}^N) + \sum_{i=1}^N \frac{\rho_i}{2}\|W_i - Z_i^k + U_i^k\|_F^2$
    for i in each layers **do**
        Update Y_i: $Y_i^{k+1} = \Pi_{S_i'}(W_i^{k+1} + V_i^k)$
        Update V_i: $V_i^{k+1} = V_i^k + W_i^{k+1} - Y_i^{k+1})$
        Apply K-mean clustering on weights and update centriod by results
    end for
end for
for k in iterations **do**
    for i in each layers **do**
        Quantize $\alpha\%$ weights to closest quantization level
    end for
    Retraining on remaining weights
end for
for j in every epochs **do**
    Retrain the centroid by results
end for
Quantizing the rest of weights

5. NEW FRAMEWORK IDEAS

The computational time in the above work is concerned since the algorithm is using lots of loops and restraining to maintain the accuracy, therefore we adjust the function in the algorithm to improve the performance. Furthermore, the compression ratio is always a direction to improve and explore and we apply additional algorithms to see if the compression ratio can be improved without destroying accuracy.

5.1 Classification with Gradients

To reduce the computational time, we try to pre-classify the redundant weights using their gradients. During the training, we pull out the gradients for each weight in every ADMM iterations as a vector, label the pruned weight as False and remaining weights as True and use them for classification. Apply support vector machine (SVM), Gaussian Naive Bayes and other classify methods to see if there is a pattern or hyperplane that classify the weights and save the computational process instead of iterative pruning with training and fine-tuning.

5.2 Parameters and Computational Time

In ADMM based algorithm, most parameters and hyper-parameters are auto-calculated and updated, such as centroids and quantization set; but few parameters are pre-defined and initialized such as pruning ratio α_i. Find the relationship between parameters, computational time, and accuracy degradation, and apply an algorithm to auto-adjust those factors to balance them.

5.3 Pruning with Layers/Neurons after ADMM

After ADMM iterative training, most of the weights are pruned and thinking in each layer, one layer has only a few neurons with unpruned weights which might not be significant in the whole model, so we can also consider that layer as a redundant layer and pruned it. Therefore, finding the layer with fewest input weights/outputs weights and pruning those layers is an idea to improve the compression ratio.

6.1 Classification with Gradients

As the backpropagation process in DNN, the gradients are significant to determine the change of weights. The Fig. 6.1 and Fig.6.2 shows the gradient changes for first two unpruned weights and first two pruned weights in the first convolutional matrix as an example, where those gradients are recorded 400 times during the training process. Few observations can be found from those figures: for pruned weights, the gradient is changing frequently with larger magnitude at the beginning and flipping around zero at the end of the training, but in unpruned weights, the gradient is changing along with whole processes with a larger-scale difference. Therefore, we based on the property and changing the path of gradients to classify the weights into two classes, pruned or unpruned weights.

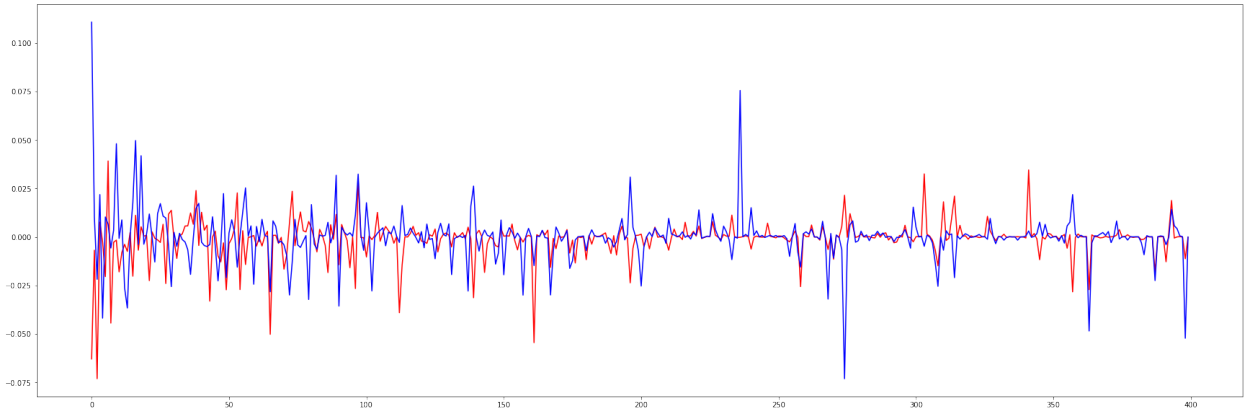


Figure 6.1: Gradient Path for Pruned Weights in first convolutional layer

There are several build-in classifiers to use, we are using SVM (support vector machine) classifier, decision tree, nearest neighbors, SGD (Stochastic gradient descent) classifier, and gaussian naive bayes classifier to test our idea. The results from LeNet-5 model with MNIST data set can be concluded in Table 6.1, which consists the number of pruned/unpruned weights, number of

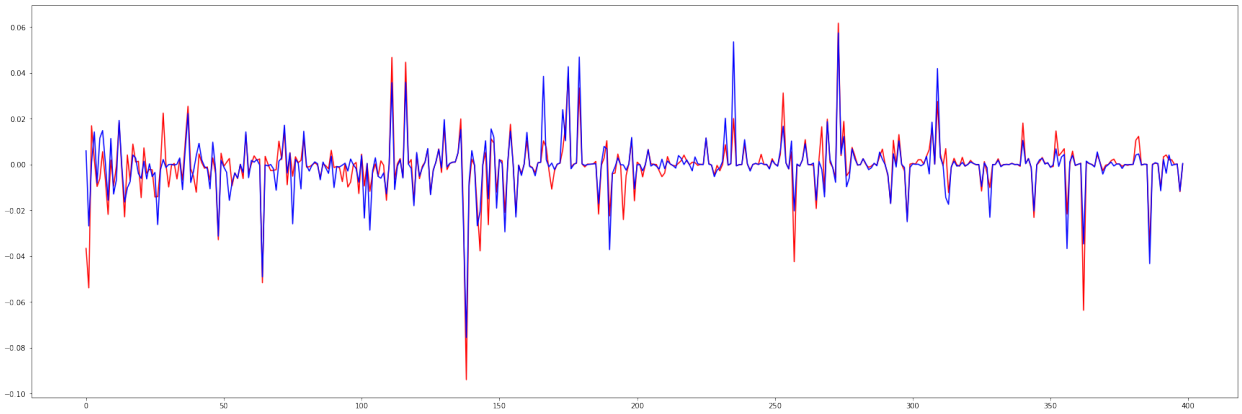


Figure 6.2: Gradient Path for Unpruned Weights in first convolutional layer

Model	Original	SVM	Decision Tree	Nearest Neighbors	SGD	Gaussian Naive
Conv 1 (Pruned/UnPruned)	400 / 100	478 / 22	365 / 135	487 / 13	287 / 213	323 / 177
(Num. of correct)		(379 / 1)	(297 / 32)	(389 / 2)	(238 / 51)	(247 / 24)
Accuracy		76.0 %	65.8 %	78.2 %	57.8 %	54.2 %
Conv 2 (Pruned/UnPruned)	23650 / 1350	23223 / 1777	19771 / 5229	24985 / 15	25000 / 0	16533 / 8467
(Num. of correct)		(22041 / 168)	(18940 / 519)	(23636 / 1)	(23650 / 0)	(16126 / 943)
Accuracy		88.84 %	77.84 %	94.55 %	94.60 %	68.27 %
FC 2 (Pruned/UnPruned)	4650 / 350	4509 / 491	4533 / 855	4999 / 1	5000 / 0	3759 / 1241
(Num. of correct)		(4197 / 38)	(3887 / 92)	(4649 / 0)	(4650 / 0)	(3556 / 147)
Accuracy		84.70 %	79.58 %	92.98 %	93.00 %	74.06 %
Total (Pruned/UnPruned)	28700 / 1800	27710 / 2790	23203 / 7297	30489 / 11	30500 / 0	22994 / 7506
(Num. of correct)		(26230 / 320)	(22009 / 606)	(28690/ 1)	(28700 / 0)	(22180 / 986)
Accuracy		87.05 %	74.15 %	94.07 %	94.10 %	75.95 %

Table 6.1: Classification results in each layers for LeNet-5 model including: number of pruned/unpruned, number of correct classify weights, and accuracy

pruned/unpruned weights in correct classes, and their accuracy for each layers and total layers. The second convolutional weights are ignored due to large numbers (400K as the number of weights) and computation complexity during the gradient pulling. Since number of pruned and unpruned weights are unbalanced, i.e. number of pruned is sixteen times larger than unpruned, the results in table is generated by auto-balance build-in function in each classifiers. Those results seem like the data is still unbalanced even using the auto-balance weight, so we tried to classify the data

using under-sampling and over-sampling algorithms but the results is similar and unchanged. As the results, the accuracy is quite low to conclude that the data is not good enough or missing some significant information to classify the weights.

6.2 Parameters and Computational Time

The parameters and hyper-parameters in the model can affect the results not only for accuracy degradation and compression ratio but also operation time. Those parameters including the pruning percentages of each layer, initial level size in the quantization model, initial cluster number set in the clustering model, etc. Within those, pruning percentiles is the most significant factor since it directly influences the number of remaining weights and test accuracy.

Two figures show the relationship of pruning percentages with accuracy and time as in Fig.6.3 and Fig.6.4 respectively. It is clear that as the pruning percentages have increased the accuracy is decreased for all layers, moreover, the second fully-connected layer is dropping moderately when the first convolutional layer is dropping rapidly. On the other hand, the computational time is increased when the pruning ratio is increasing for all cases, and the convolutional layers' pattern is more likely to each other as well as fully-connected layers, as shown in Fig.6.5.

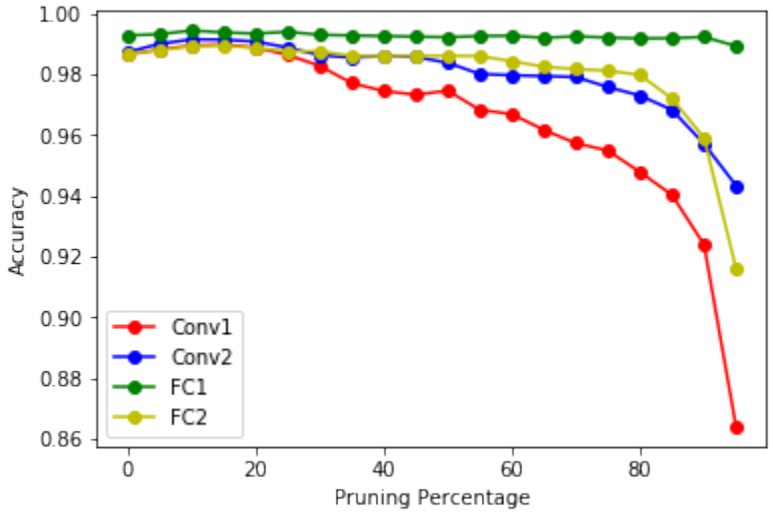


Figure 6.3: Pruning Percent versus Accuracy

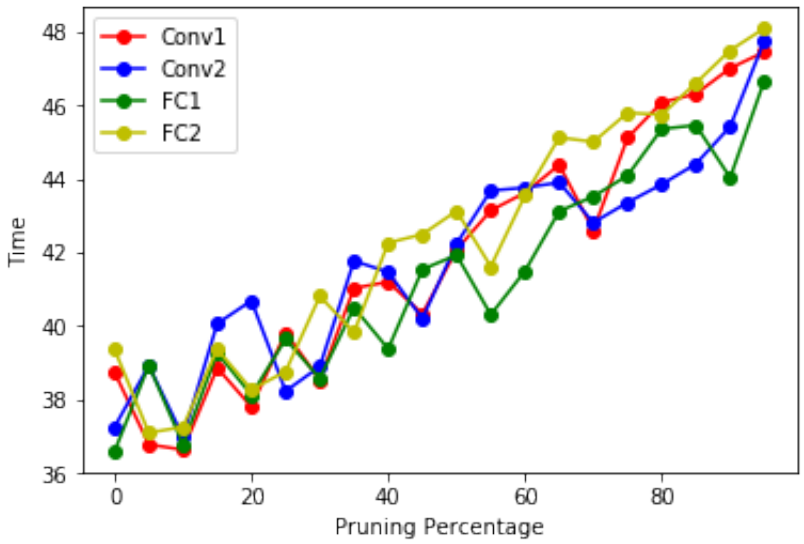
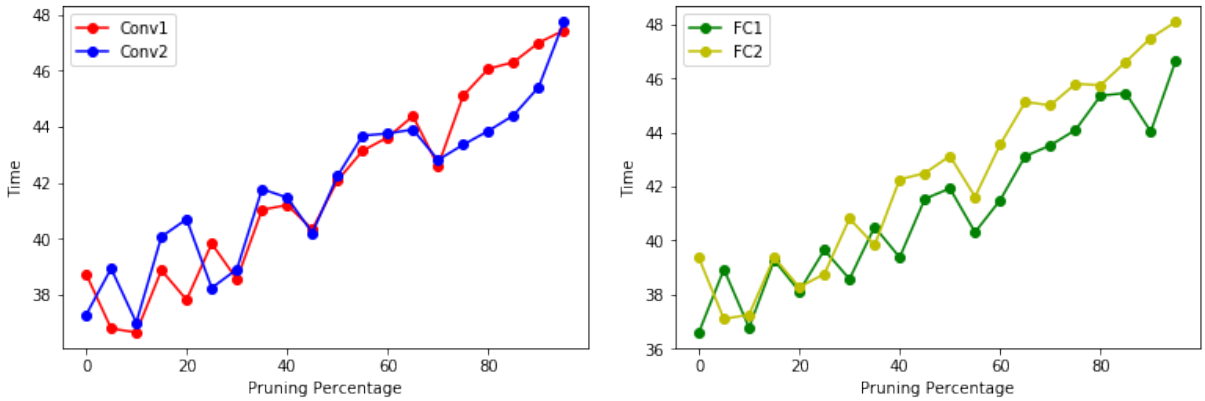


Figure 6.4: Pruning Percent versus Execution Time.

As setting the auto-adjusted parameters, the combination of pruning percentages also needs to be considered. We set the inner loop for each layer, given a certain range of pruning ratio and increase their ratio by steps to find the best accuracy. The exhausting search and binary search are applied as our searching algorithms for the best combination of α_i. The results are compared in the Table.6.2, includes the setting range, steps, and related accuracy. It can be noticed that although the accuracy is similar to the manual adjusted (as [15]) but the number of weights has a large difference, and the reason is due to the sensitivity between accuracy and pruning ratio. From Fig. 6.3 we can see that only small amount of accuracy is dropped at most 2% while the pruning ratio is less than 80, which means if given the certain allowable accuracy degradation then the maximum pruning ratio can be found.

	P1	P2	P3	P4	Num of Weights	Accuracy
ADMM (Ye, 2018)	80	94.6	99.8	93	2.58 K	98.6 %
Auto-Adjusted (Range, Step)	80 (80-100, 1)	99.6 (95-100, 0.2)	95 (95-100, 1)	93 (90-100, 1)	20.55 K	98 %

Table 6.2: Comparison Auto-adjustment in Pruning Ratio on LeNet-5



(a) Comparison within Convolutional layers

(b) Comparison within FC layers

Figure 6.5: Detail Comparison of Pruning Percentage vs. Time

The largest problem for this algorithm is the time-consuming problem, where the exhausted search and binary search require a large computational time, also the balance between accuracy and pruning ratio is another problem. Therefore, we can conclude that although the model can be auto-set the parameters, the performance in accuracy and pruning ratio is not as good as expected, and the problem of how efficiency for accuracy and the remaining number of weights should also be considered.

6.3 Pruning with Layers

To compress deeply in weight numbers and computational time, the training model can be adjusted by reducing the layers. Since most of the weights are pruned in the pruning and fine-tuning method, less than ten percent of neurons are leftover. Thinking about the weight numbers in each layer, e.g. first fully-connected layer only has 800 out of 400K weights left after pruning, some layers or kernels might not significant in the whole model although they have unpruned weights. On the other hand, looking back to the Fig.6.3, fist fully-connected layer's accuracy is mostly unchanged under all pruning percentage, in other words, the weights in first fully-connected layer is insignificant in the model and we can get rid of the whole layer to reduce the complexity of the model when training.

Implementing the idea into the ADMM based model, the simple LeNet-5 model is a good starting point along with the MNIST digit recognition dataset as training. As using the MNIST data set, the input is given as figures of hand-written digits which are decomposed as 32X32 grids, and the output is classified as 10 classes represent the different digits. The LeNet-5 architecture consists the training process as: input, first convolutional layer, ReLU (rectified linear unit) activation function layer, max-pooling layer, second convolutional layer, ReLU layer, max-pooling layer, first fully-connected layer, dropout layer to prevent the co-adaptation of features and control the complexity of model, ReLU layer, second fully-connected layer, and output classifiers.

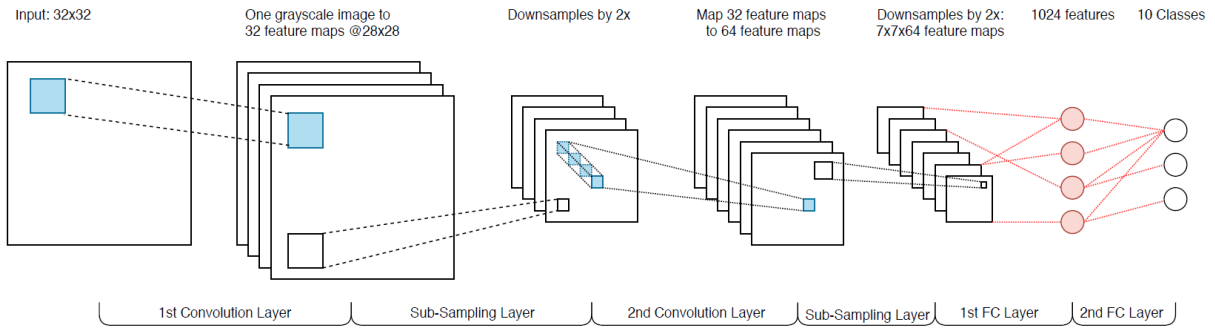


Figure 6.6: Model of LeNet-5: including two convolutional layers, two sampling layers, and two fully connected layers.

As the conclusion before, the new model is adjusted as pruning out the first layer, in which the output of the second convolutional layer is not connected to the first fully-connected layer but directed connected with the second fully-connected layer. The new model is showing as Fig.6.7, which consists of the architecture as following: input, first convolutional layer, ReLU layer, max-pooling layer, second convolutional layer, ReLU layer, max-pooling layer, dropout layer, fully-connected layer, and output classifiers.

This model is applied after the pruning and fine-tuning methods. First, using the original LeNet-5 model to train, prune and fine-tune the model with ADMM based framework, then the remaining weights or neurons are those significant connections in the model. Second, combining

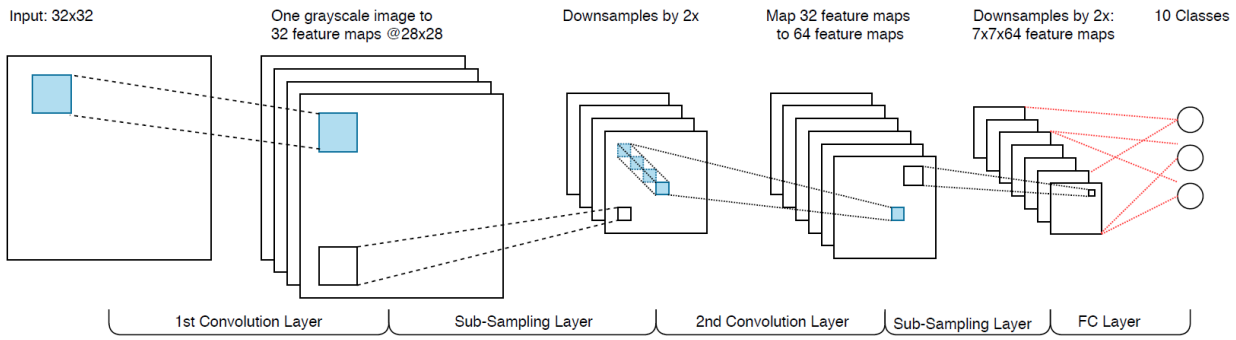


Figure 6.7: New model from LeNet-5: including two convolutional layers, two sampling layers, and one fully connected layers.

two fully-connected weights matrix into one: finding out which neurons in fully-connected layer has non-zero weights, taking the non-zero weights from left side (first fully-connected weight which is taken from convolutional layer to that neurons) and the weights from right side (second fully-connected weight which is taken from neurons to the output), using mathematical expression to combine them and generate the new one matrix to new model. Last, applying this new fully-connected weight matrix to the new model (with original two convolutional weight matrix), setting a new pruning percentage for this layer and prune/fine-tune the model to get the final weights. The process can be concluded as algorithm 3.

ADMM Method (Ye, 2018)	Layer	Number of Weights	Number of Weights after Pruning	Pruning Percentage	Weight Bits	Accuracy
	Conv 1	0.5 K	0.1 K	20 %	5 (25)	-
	Conv 2	25 K	1.35 K	5.4 %	3 (8)	-
	FC 1	400 K	0.8 K	0.2 %	2 (4)	-
	FC 2	5 K	0.35 K	7 %	3 (6)	-
	Total	430.5 K	2.58 K	0.6 %		98.67 %
New Method	Conv 1	0.5 K	0.1 K	20 %	5 (25)	-
	Conv 2	25 K	1.35 K	5.4 %	3 (8)	-
	FC	405 K	0.482 K	0.12 %	2 (3)	-
	Total	430.5 K	1.932 K	0.45 %		98.43 %

Table 6.3: Comparison in Layer-Wise Weight Pruning Results on LeNet-5

Algorithm 3 Weight Pruning Method with Layers

Input: Data after regular Weight Pruning
Initialization: Determine the new pruning percentage α_i for fully-connected layer in new model
Find the index of neurons with non-zero weights (both sides of fully-connected matrix)
for i in left index of non-zero weights (out of 800) **do**
  for j in right index of non-zero weights (out of 500) **do**
    for k in output (10 classes) **do**
      new weight $= (FC1[i][j] + FC1_{bias}[j]) \cdot FC2[j][k]$
    end for
  end for
end for
Prune the new fully-connected weight with new pruning range α_i
for j in every epochs **do**
  Retrain the weights for new model and weights
end for

This method brings better results in the weight numbers. In the original method [15], the total number of weights in the fully-connected layer after pruning is 1.15K where 0.8K from the first fully-connected layer and 0.35K from the second fully-connected layer. Using our new method the total number of weights reduces to only 0.482K by applying the additional pruning parameter $\alpha = 80$. The total number is reduced from 0.6% to 0.45% and the computational time is also decreased since the complexity of the model is reduced by less than a fully-connected layer. The adverse effect of the model is the accuracy is degradation, the original accuracy after pruning is 98.67% but the accuracy after applying the new model is dropped to 98.43%, where the degradation is 0.24%.

7. CONCLUSION AND FUTURE WORKS

The final comparison of results can be concluded as Table. 7.1, including the baseline of LeNet-5 model, iterative pruning method [9], ADMM based framework [15], and our pruning layers method. The significant improvement is the number of weights in each layer, which has 1.932K out of 430.5K. But the accuracy in this model is decreased by more than 0.2% which can be improved in future works.

Model	**LeNet-5 Baseline**	**Iterative Pruning (Han, 2016)**	**ADMM (Clustering) (Ye, 2018)**	**ADMM (Quant.)**	**New Method (Clustering)**
Accuracy Degration	0.0 %	0.1 %	0.1 %	0.2%	0.2%
Number of Weights	430.5 K	35.8 K	2.57 K	2.57 K	1.932 K
Conv Weight Bits	32	8	3	3	3
FC Weight Bits	32	5	2/3	2/3	2

Table 7.1: Compression final results for different model using LeNet-5 for MNIST data set.

REFERENCES

[1] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, pp. 541–551, Dec 1989.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, (USA), pp. 1097–1105, Curran Associates Inc., 2012.

[3] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "Model compression and acceleration for deep neural networks: The principles, progress, and challenges," *IEEE Signal Processing Magazine*, vol. 35, pp. 126–136, Jan 2018.

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

[5] H. Babak, S. David G., W. Gregory, and W. Takahiro, "Optimal brain surgeon: Extensions and performance comparisons," in *Proceedings of the 6th International Conference on Neural Information Processing Systems*, NIPS'93, (San Francisco, CA, USA), pp. 263–270, Morgan Kaufmann Publishers Inc., 1993.

[6] R. Reed, "Pruning algorithms-a survey," *IEEE transactions on Neural Networks*, vol. 4, pp. 740–747, Sep 1993.

[7] J. M. Alvarez and M. Salzmann, "Compression-aware training of deep networks," *CoRR*, vol. abs/1711.02638, 2017.

[8] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient transfer learning," *CoRR*, vol. abs/1611.06440, 2016.

[9] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," *CoRR*, vol. abs/1510.00149, 2015.

[10] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," *CoRR*, vol. abs/1506.02626, 2015.

[11] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," *CoRR*, vol. abs/1504.04788, 2015.

[12] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *CoRR*, vol. abs/1608.08710, 2016.

[13] F. Tung and G. Mori, "Deep neural network compression by in-parallel pruning-quantization," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2018.

[14] T. Zhang, S. Ye, K. Zhang, J. Tang, W. Wen, M. Fardad, and Y. Wang, "A systematic DNN weight pruning framework using alternating direction method of multipliers," *CoRR*, vol. abs/1804.03294, 2018.

[15] S. Ye, T. Zhang, K. Zhang, J. Li, J. Xie, Y. Liang, S. Liu, X. Lin, and Y. Wang, "A unified framework of DNN weight pruning and weight clustering/quantization using ADMM," *CoRR*, vol. abs/1811.01907, 2018.

[16] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Found. Trends Mach. Learn.*, vol. 3, pp. 1–122, Jan. 2011.