Optimizing CNNs for Embedded Systems

V. Naveen Chander, Tejas Oturkar

Abstract

The increasing success of Convolutional Neural Networks (CNNs) is accompanied with a significant increase in the computation and parameter storage costs which inhibits their porting to embedded systems. We propose an optimization flow for optimization CNNs. Structural pruning of neural network parameters reduces computation, energy, and memory transfer costs during inference. In this mini project, we study a method to estimate the importance of all network parameters in a computationally efficient manner using Taylor Series based approximation. The pruning technique is validated on a CNN and a multi-layer perceptron neural network for MNIST dataset. The pruned neural network is quantized with multiplier-less optimization technique and is synthesized on FPGA accelerator to demonstrate the computational advantages over the original neural network.

1. Introduction

CNNs are largely over-parameterized (*Du* et al. 2019*)*, and require much computation during inference. Several algorithms have been employed to reduce the complexity of CNNs. These include Pruning, Weight Sharing, Low rank approximation and quantization. Among these, parameter pruning is done as the first and foremost step towards model optimization, which aims at eliminating redundant model parameters with tolerable performance degradation. Pruning typically reduces the model size by around 70%-80%. DNNs with low precision data formats have enormous potential for reducing hardware complexity (Tann et.al, 2017). Thus, the pruned model's weights are quantized to values that are powers of two so that all multiplications are reduced to meagre left-shift operations, thereby reducing the inference-latency by factor of 6 and the hardware resource utilization by 1/3rd. All this comes at the cost of reduction in the accuracy. Here, we show that a considerable level of optimization can be performed before the accuracy starts to degrade significantly.

2. Model Pruning Flow

Neural Network Pruning entails taking an input model as $f(x; W)$ and producing a new model $f(x; M \odot W')$, where, W' is a subset set of parameters W of the neural network as $f(x; W)$, which will be zeroed out using a binary mask $M \epsilon \{0,1\}^{|W'|}$.

Many techniques in the literature rely on the belief that the magnitude of a weight and its importance are strongly correlated. However, this relation is empirical and does not yield promising results as CNNs scale in size.(Liu et al., 2019) A more reliable method would be to frame a pruning criterion that iteratively removes the least important set of neurons depending upon their contribution to the final output of the neural network.(Molchanov et al., 2019)

In this course project, we propose to formulate the optimization problem with pruning criteria from the paper [1] and use formulation in [Song Han et.al 2015] to regain the accuracy lost while pruning as shown in the algorithm below.

Algorithm 1 Pruning and Fine-Tuning

Input: N, the number of iterations of pruning, and X, the dataset on which to train and fine-tune

1: $W \leftarrow initialize()$
2: $W \leftarrow trainToConvergence(f(X; W))$
3: $M \leftarrow 1^{|W|}$
4: **for** i in 1 to N **do**
5: $\quad M \leftarrow prune(M, score(W))$
6: $\quad W \leftarrow fineTune(f(X; M \odot W))$
7: **end for**
8: **return** M, W

3. Literature Review

Pruning can be broadly classified into two categories – Pruning of individual weights (un-structured pruning) and pruning of filter networks (structured pruning).

Unstructured methods started in 1990s with methods like Optimal Brain Surgeon (Hassibi & Stork, 1993) and Optimal Brain Damage (LeCun et al., 1990) which pruned weights based on second derivative of a Cost function for shallow CNNs. This method ceased to be effective after the inception of DNNs in 2012. (Han. et. al 2015) proposed a pruning pipeline based on magnitude pruning of weights to demonstrate a model compression of up to 9x on VGG-16 new without appreciable loss in accuracy. While unstructured methods are highly effective in compressing models, they are empirical methods without a proper mathematical framework and do not demonstrate the same level of effectiveness in large CNNs. (Liu et al., 2019). Moreover, they involve a lot of hyper-parameters such as layer pruning threshold etc., which will vary with neural network and the dataset, thus affecting the reproducibility of technique.(Dong et al., 2017) Another major drawback of unstructured pruning technique is disruption of regularity of a network, especially in CNNs, by partially pruning filters etc., Thus, modern hardware can no longer exploit regularities in CNNs, thus imposing requirements for special hardware or sparse BLAS libraries. (Han et al., 2016)

Structured Methods propose target pruning convolutional filters or whole layers as against individual weights (Dong and Yang, 2019). A saliency score based on ℓ_1 *or* ℓ_2 norm of

the Kernel Matrix can be set as pruning criterion (Li et al., 2017) and a fixed number of channels in each convolutional layer can be pruned depending on these scores. As the scores are adaptively computed with respect to the model as well as the task, these methods can be easily applied to different scenarios (Chen et.al.,2019). However, the drawback is that this method too tends to associate the pruning criteria mainly on the magnitude of the weight, which is an empirical criterion and can have problems as mentioned before.

Weight sharing techniques (Babu et al., 2015), device technique to remove the neurons rather than weights. Also, to compensate for the information embedded int the lost connections, pruned weights are added to the existing neural connections. This technique is effective for the FC layers of CNNs. Another structured pruning technique called, Self-adaptive pruning technique (Chen et al., 2019) considers the fact that the inputs together with the weights determine the extent of neural activation. Thus, the weights are pruned selectively based on the input values in real time. However, it comes at an expense of building a saliency estimator function for each layer to take decisions in real time.

Quantization of neural networks has been proposed by (Courbariaus et al., 2016), (Ghasemzadeh ret al., 2018) etc., involves quantizing all the weights to binary values, $W \epsilon \{-1, 1\}$. However, does not apply well to already pruned networks since, a high weight count is required to achieve a certain accuracy from a Binary Neural Network. Tann, et al. (2017) propose a multiplier-less neural network architecture wherein all the network weights are rounded off to their nearest powers of 2, which worked well on our pruned neural network.

Shupeng Gui et al. (2020) define a framework for unifying the pruning and quantization constraints into a single optimization problem. However, they do not use a structured pruning. Molchanov et.al (2019) define a framework to set a pruning criterion based on the importance of a channel. Our work develops a unified framework based on pruning criteria set by Molchanov et.al (2019) and the quantization technique by [Tann et al., 2017].

4. Problem Formulation

Given neural network parameters $\boldsymbol{W} = \{w_0, w_1, \dots, w_M\}$ and a dataset $D = (x_0, y_0), (x_1, y_1), \dots, (x_K, y_K)$ composed of inputs (x_i) and outputs (y_i) pairs. The task of training is to minimize the error E by solving:

$$\min_{\boldsymbol{W}} E(D, \boldsymbol{W}) = \min_{\boldsymbol{W}} E(y|x, \boldsymbol{W}) \qquad (1)$$

In case of Pruning we can include a sparsification term in the cost function to minimize the size of the model:

$$\min_{\boldsymbol{W}} E(D, \boldsymbol{W}) + \lambda \|\boldsymbol{W}\|_{\boldsymbol{0}} \quad (2)$$

where, λ is the scaling coefficient and $\|\cdot\|_0$ is the ℓ_0 norm which represents the number of non-zero elements.

But, (2) Is non-convex, NP-hard and requires Combinatorial Search for Minimization. Alternatively, we perform the usual Optimization, without the sparsity constraint in (2) to get the full set of parameters $\boldsymbol{W}$ upon convergence of the original optimization and then gradually, reduce this set W by a few parameters at a time.

Define *Importance* $(\mathcal{I}_m)$ as the squared change in loss induced by removing a specific filter from the network.

$$\mathcal{I}_{m} = (E(D, \boldsymbol{W}) - E(D, W|w_m = 0))^2 \qquad (3)$$

Since computing the exact importance is extremely expensive for large networks, it can be approximated with its Taylor Series Expansion in the vicinity of $\boldsymbol{W}$ as :

$$\mathcal{I}_m^{(2)}(W) = \left(g_m w_m - \frac{1}{2} w_m \boldsymbol{H}_m W\right)^2 \qquad (4)$$

where $g_m = \frac{\delta E}{\delta w_m}$ are elements of the gradient $\boldsymbol{g}$, $H_{i,j} = \frac{\delta^2 E}{\delta w_i w_j}$ are the elements of the Hessian $\boldsymbol{H}$, and $\boldsymbol{H}_m$ is the m^{th} row of $\boldsymbol{H}$. We can get even more compact expression using the First order Taylor Series which gives:

$$\mathcal{I}_m^{(1)}(W) = (g_m w_m)^2 \qquad (5)$$

To approximate the joint importance of a structural set of parameters $\boldsymbol{W_S}$ e.g., in convolutional filter, we have two alternatives. We can define it as a *group contribution:*

$$\mathcal{I}_{\mathcal{S}}^{(1)} \triangleq \left(\sum_{s \epsilon \mathcal{S}} g_s w_s\right)^2 \dots (6)$$

5. Results

We demonstrate importance pruning method on CNN and MLP topologies. Subsequently, the pruned architecture is quantized with multiplier-less optimization and made ready for running on an FPGA based hardware accelerator.

a. CNN Results: Method of importance function pruning was carried out on 7-layer CNN consisting of two convolutional layers namely Conv 1 and Conv 2.

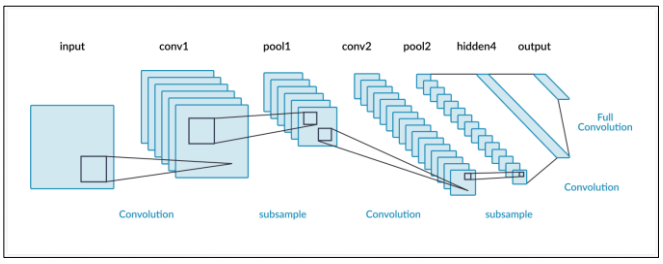


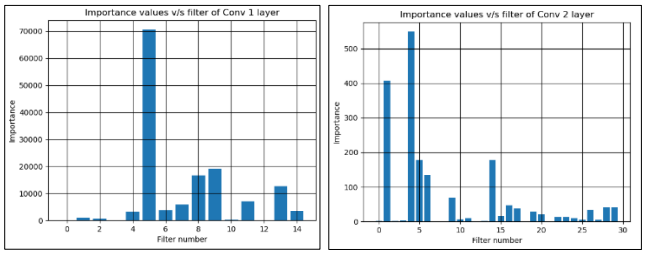
Figure 1 Standard Convolutional neural network used in our problem

We train the original CNN shown above on MNIST handwritten digits dataset *ADAM* optimizer to obtain 99.7% accuracy on test dataset.We jointly approximate the importance of parameters of each filter of first convolutional layer (Conv 1). Say, Conv 1 has total 'F' filters from (6) importance of p^{th} filter in *Conv 1* layer is approximated as,

$$\mathcal{I}_p^{(1)}(W) \sim (\textstyle\sum_{a=1}^{n} g_a w_a)^2 \qquad (7)$$

where, p denotes filter number in Conv 1 layer, g_a denotes gradient of each parameter of the p^{th} filter w.r.t cost function (obtained using AutoGrad function in tensorflow). w_a denotes the filter weight of p^{th} filter having n weights.

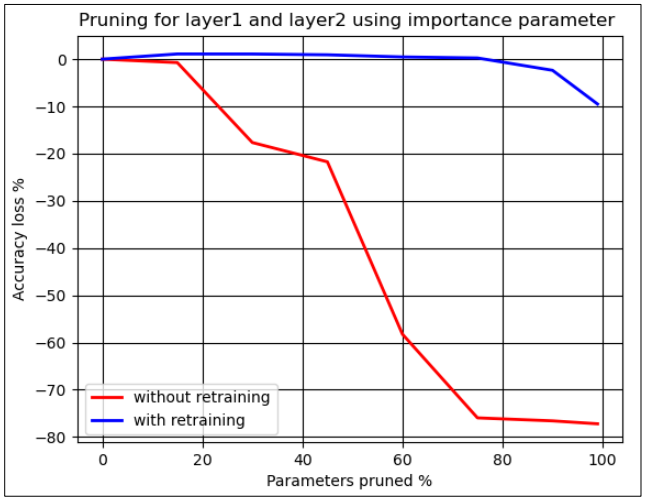
Importance of each filter is calculated by squaring the sum of product of each weight and its gradient belonging to that filter. Following figure shows importance of each filter in *Conv 1* layer.



From importance graph it can be observed that filter-1, filter-4, filter-5, filter 14 of Conv 2 generate feature maps that are most relevant for classification of data.

Implementing Pruning and Retraining Iteration: We prune all filters layer-wise, with importance than *1%* (hyper-parameter) of the maximum importance. Check the accuracy on test dataset. Retrain the pruned network and evaluate it. This increases the accuracy. Do this iteratively till until *99 %* of the filters in both convolutional layers were pruned.

Pruning Results: A plot showing the effect of pruning on accuracy is shown below.

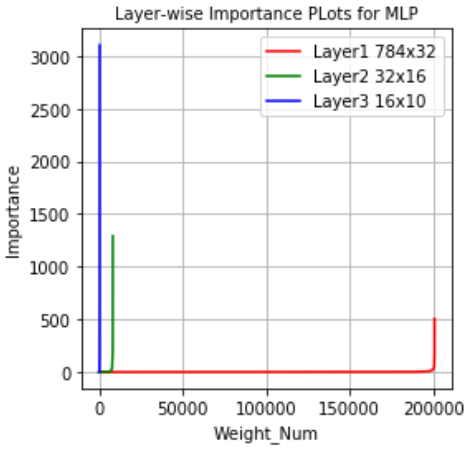


b. Multi-layer Perceptron Results

The following MLP configuration was found to yield the best accuracy for MNIST dataset.

Figure 2 MLP Architecture

The model has a total of 209514 trainable parameters. The model was trained in tensorflow using Adam optimizer to obtain an accuracy of 99.38% on training set and 97.62% on test set. Importance was calculated for all weights. Layer-wise sorted importance values are plotted as shown.



Observations: Importance values decrease from Layer3 to Layer1. The relatively low importance of the first hidden layer neurons and weights when compared to those in the last layer allow us to prune a larger percentage of weights from the first hidden layer. In the first hidden layer, most of the weights have relative importance close to zero and a clear set of weights have high importance values.

Pruning Criteria: Thus, the pruning criterion for the first layer can be set to prune all the weights that have importance **< 0.5%** of the maximum importance in first layer. Here, 0.5% is the Hyper-parameter. Second and Third layers shall not be pruned.

Pruning Results: On pruning, the number of parameters in the first layer comes out to be = 16485. Thus, number of neurons required in the first hidden layer to achieve this neuron count is close to 32. Thus, the pruned network is **784→32→16→10** Number of Model Parameters: **25120**

Hardware Results Here, we discuss about the performance at the hardware level. The pruned MLP was quantized to 8-bits and multiplier less optimization was applied to eliminate the need for hardware multipliers. The MLP was exported to (Zynq-7000) FPGA based hardware accelerator and the hardware performance of the optimized and unoptimized networks were evaluated:

Table 1 Hardware Results

Parameter	**Original MLP**	**Optimized MLP**
Latency (ms)	23.06	0.52
Mem Used (KB)	9600	432

Thus, we witness a significant computational advantage with the optimized MLP.

7. References

Molchanov et al., *Importance Estimation for Neural Network Pruning* , 2019

Shepung Gui et al., Model Compression with adversarial robustness, 2019

Song Han, *Efficient Methods and Hardware for Deep More Learning, PhD Thesis,* 2017

Yeom et al., *Pruning by Explanation: A Novel Criterion for Deep Neural Network Pruning,* 2019"

Mengqing Wang, *Pruning Convolutional Filters with First Order Taylor Series Ranking,* 2016

Chen et al., *Self-Adaptive Pruning,* 2019

Ehud Karnin., *A Simple procedure for training Back propagation trained Neural Networks,* 1990

Yu Cheng et al., *A Aruvey of Modern Techniques and Acceleration of Deep Neural Networks,* 2019

Wang et al., Structured *Pruning for ConvNets via Incremental Regularization,* 2019

Han et al., *Learning Both Weights and Connection for Efficient Neural Networks,* 2016.

Suraj Srinivas and Venkatesh Babu, *Data-Free parameter Pruning for Deep Neural Networks,* 2015

Wang et al., *Pruning from Scratch*, 2019

Liu et al., *Rethinking the Value of Pruning,* 2019

Du et al., *Optimizing Convolutional Network Accelerator*, 2016

Y. Gong, L. Liu, M. Yang, and L. D. Bourdev, *Compressing deep convolutional networks using vector quantization,* 2014

Y. W. Q. H. Jiaxiang Wu, Cong Leng and J. Cheng, *Quantized convolutional neural networks for mobile devices*, 2016

S. Han, H. Mao, and W. J. Dally, *Deep compression: Compressing*

deep neural networks with pruning, trained quantization and Huffman coding, 2016

B. Hassibi, D. G. Stork, and S. C. R. Com, *Second order derivatives for network pruning: Optimal brain surgeon,* 1993

Li et al., *Pruning Filters For Efficient ConvNets,* 2017

Courbarriux et al., "*Binarized Neural Networks: training Neural Networks with Weights and Activations Constrained to +1 or -1",* 2016

Ghasemzedash et al., *"ReBNet: Residual Binarized Neural Network",* 2018

Tann et al., *"Hardware-Software Codesign of Accurate, Multiplier-free Deep Neural Networks",* 2017

Sarwar et al., "*Multiplier-less Artifical Neurons Exploiting Error Resiliency",*

He et al., *"Channel Pruning for Accelerating Very Deep Neural Networks"*

Yu et al., *"NISP: Pruning Neural Networks using Neuron Importance Score Propagation"*

Park et al., *Look ahead: Far-sighted Alternative of Magnitude based Pruning*, 2020

Molchanov et al., *Pruning Convolutional Neural Networks for resource efficient inference,* 2017

Dong and Yang, *Network Pruning via Transformable Architecture Search*, NeurIPS-2019

Machine Learning Lectures, *CS229*

Deep Learning Lectures, *CS231*