Combining Weight Pruning and Knowledge Distillation For CNN Compression

Nima Aghli  
Florida Institute of Technology  
150 W University Blvd, Melbourne, FL 32901  
naghi2014@my.fit.edu

Eraldo Ribeiro  
Florida Institute of Technology  
150 W University Blvd, Melbourne, FL 32901  
eribeiro@fit.edu

Abstract

Complex deep convolutional neural networks such as ResNet require expensive hardware such as powerful GPUs to achieve real-time performance. This problem is critical for applications that run on low-end embedded GPU or CPU systems with limited resources. As a result, model compression for deep neural networks becomes an important research topic. Popular compression methods such as weight pruning remove redundant neurons from the CNN without affecting the network’s output accuracy. While these pruning methods work well on simple networks such as VGG or AlexNet, they are not suitable for compressing current state-of-the-art networks such as ResNets because of these networks’ complex architectures with dimensionality dependencies. This dependency results in filter pruning breaking the structure of ResNets leading to an untrainable network. In this paper, we first use the weight pruning method only on a selective number of layers in the ResNet architecture to avoid breaking the network structure. Second, we introduce a knowledge distillation architecture and a loss function to compress the untouched layers during the pruning. We test our method on both image-based regression and classification networks for head-pose estimation and image classification. Our compression method reduces the models’ size significantly while maintaining the accuracy very close to the baseline model.

1. Introduction

Deep-learning algorithms are currently the major producer of state-of-the-art results for computer-vision problems such as object detection [22, 23, 24], image classification [13, 9], and image segmentation [4, 5].

Despite their remarkable accuracy, most state-of-the-art CNNs are both computationally expensive and memory demanding. For instance, deep CNN architectures such as ResNets [9] have millions of parameters that require expensive hardware such as GPUs with a large amount of memory and parallel-computation capabilities if real-time inference is expected.

Different methods have been proposed to reduce the computational complexity of CNNs while maintaining the compressed network’s accuracy similar to that of the original model. Leng et al. [20] proposes a weight quantization algorithm to compress CNNs by quantizing each full-precision weight in the network to a small number of bits. Hinton et al. [10] introduces the teacher-student knowledge distillation (KD) method, where a deeper teacher network distills the knowledge of its feature maps to a smaller student network. Additionally, different weight-pruning methods have been proposed [12, 8], where neurons that do not contribute to the model’s prediction are removed.

In this paper, we propose a network-compression method for deep CNNs by combining weight pruning via activation analysis [12] and knowledge distillation. To demonstrate our approach, we apply the compression method to a head-pose estimation regression network and to an image-classification network without any significant loss of accuracy over the original uncompressed network. We also compare inference time of the compressed networks on a PC, embedded GPUs, and embedded ARM CPU. Finally, we compare the compressed head-pose model’s robustness against occlusion, motion-blur, and brightness changes.

2. Related Work

Early and shallow networks have been largely outperformed by deeper and wider networks with complex architectures as these networks can capture more complex features. However, most deeper networks suffer from significant redundancy [6] as many network neurons have no contribution to the prediction while still consuming memory and computation. Such redundancy can be reduced by pruning neurons based on their numerical properties. In general, network trimming methods are divided into connection and weight-pruning categories. Han et al. [8] propose a model-compression method as an iterative approach that prunes the connections from the weights with near-zero values followed by weight quantization. Instead of pruning connections, Hu et al. [12] propose an iterative neuron pruning
as they show that connection pruning does not bring large improvements on GPUs since convolutional operations in GPUs are converted from 2-D to 1-D vectors followed by matrix multiplication. Hence, when pruning connections instead of neurons, the multiplications stay the same.

Knowledge distillation is another approach to compression that transfers useful feature representation from a teacher network to a student network that has less parameters, and then fine-tunes the student network after knowledge distillation. The teacher-student knowledge-distillation method was first proposed by Hinton et al. [10] for classification networks by introducing a distillation loss that uses the softened output of the softmax layer in the teacher network. One of the main challenges with the proposed method was its reduced performance when applied to very deep networks. Additionally, the proposed softened softmax loss was only applicable to classification tasks. To address these issues, Romero et al. [25] used an intermediate representation of the teacher model as a hint in addition to the output layer, which improved performance when distilling knowledge from deeper teacher networks. Yim et al. [32] applied knowledge distillation to the ResNet architecture by minimizing the L2 loss of Gramian [7] feature matrix in the ResNet modules between teacher and student.

Currently, proposed weight-pruning methods are applied to basic CNN architectures such as AlexNet [19] and VGG [27]. The deeper state-of-the-art models, such as ResNets [9], have complex architectures that limit the application of pruning methods on them. In this paper, we focus on compressing ResNet architectures as they are vastly used in current state-of-the-art classification and regression computer-vision tasks. Our compression method applies to all ResNet-based models.

Our main contributions are threefold: (1) We propose a new weight-pruning strategy for the ResNet architecture inspired by zero-activation pruning [12]. (2) We propose a new knowledge-distillation architecture by using the pruned model as a teacher. (3) We introduce a new distillation loss to transfer knowledge from the teacher to the student model. Finally, we validate our proposed method by compressing ResNet-based image regression and classification networks.

3. Method

Our method has two main steps: 1) Pruning the baseline network by activation analysis to remove neurons that do not contribute to prediction output. 2) Performing knowledge-distillation from the pruned (teacher) network to a smaller (student) network to achieve further compression.

3.1. Pruning ResNets with zero activation analysis

As our focus is the real-time inference on GPUs, neuron pruning has shown effectiveness in reducing GPU computation, we have employed the iterative weight-pruning method from [12] with some differences. First, Hu et al. [12] applies their method to relatively simple and shallow networks such as VGG-16 [27] and LeNet [19]. Instead, we apply pruning to deeper and more complex ResNet architecture. The main challenge with pruning neurons from the ResNets is the dimensionality dependency between some layers to the layers in the previous residual blocks. Residual blocks are the main building parts of ResNets and consist of a residual connection from the previous residual block to the last layer of the current block, followed by an add operation between them. As a result, pruning any layers with such a dependency results in each side of the add operation to have different dimensions and hence, making the model untrainable. Figure 1 shows the internal architecture of two different residual blocks with prunable and un-prunable layers colored in yellow and red, respectively.

Second, Hu et al. [12] apply pruning to a small number of layers in the model as they point out that pruning too many neurons from multiple layers at one step will reduce the model’s performance so much that it cannot be recovered with fine-tuning. In this paper, we show that by skipping unprunable layers in the pruning process, we can remove a large number of filters from multiple layers where the fine-tuning step can recover the model’s accuracy on the validation-set to be close to that of the baseline model.

To calculate the importance of any neuron in prunable layers, we calculate the Average Percentage of Zeros (APoZ) [12] for activation output of each neuron for all the samples in the validation-set. Given a prunable layer $PL_i = \{i = 1, 2, 3, \ldots, n\}$, where $n$ is the total number of prunable layers in the network, we calculate the importance
of the convolutional filter $c$ in layer $i$ as follows:

$$PL_c^{(i)} = \frac{\sum_{k=1}^{N} \sum_{j=1}^{M} f(PL_{cj}^{(i)}(k))}{N \times M},$$  \hspace{1cm} (1)$$

where $M$ and $N$ are the total number of validation samples and dimension of the output feature map in the channel $c$ respectively and $f$ is calculated as follows:

$$f(\cdot) = \begin{cases} 1, & \text{if } f(PL_{cj}^{(i)}(k)) = 1 \\ 0, & \text{if } f(PL_{cj}^{(i)}(k)) = 0. \end{cases} \hspace{1cm} (2)$$

The pruning process starts by calculating neuron’s importance in all prunable layers using Equation 1. If the APoZ of the neuron is larger than the standard deviation of the average APoZs in the same layer, both the neuron and its connections are removed. Once the pruning of all layers is completed, the network is fine-tuned to recover the original accuracy using the baseline model’s training configuration. We repeat the process until the network accuracy drops significantly after the iteration.

### 3.1.1 Knowledge distillation from pruned network to student

One of the challenges with the current KD methods is that the number of layers and neurons in the student network is selected arbitrarily. In other words, it is not guaranteed that the student network will not have redundancy in the layers after KD. As a result, the student network can not be considered entirely compressed. We use the pruned model as a teacher instead of the vanilla model in our proposed distillation method. This way, we can ensure that the distillation method will not transfer the student model’s possible redundancies. However, as the pruning method could not prune neurons from the un-prunable layers, we reduce the size of those layers in the distillation process by reducing them by a ratio in the student network. As a result, the number of parameters in the student network drops significantly compared to the teacher network.

KD methods that use intermediate layer representation tend to distill the knowledge from the first and the last section of the residual block [7] or use the teacher’s middle-layer as a hint [25]. In practice, deeper layers in the network learn the complex features, where the shallow layers learn the simple features. Since the complex features are hard to learn, we distill layer-wise knowledge from the teacher network’s last pruned layers and let the student learn the simple features during the KD process. Our proposed approach teaches the student network to minimize the cosine similarity between deep layers and the prediction layer with respect to the teacher’s layers. Figure 3 shows the complete architecture of our proposed KD method on ResNet50 network for head-pose estimation.

To distill the knowledge of the teacher’s layers to the student’s layers, we propose a new loss function. Given a teacher network, $T$, and student, $S$, we minimize the following distillation loss:

$$CDL = \sum_{i=1}^{M} \frac{T_i \cdot \hat{S}_i}{||T_i|| \cdot ||\hat{S}_i||} + \lambda \frac{T_o \cdot \hat{S}_o}{||T_o|| \cdot ||\hat{S}_o||},$$ \hspace{1cm} (3)$$

where $M$ is the number of the intermediate KD layers in the teacher network and $T_o$ and $\hat{S}_o$ is the output layer of the network. For regression network the output vector is regression layer and for classification is the softmax layer. $\lambda$ is a hyper-parameter to define the importance of the final prediction in last layer over the total loss. In this paper, we set $M$ to 9 and $\lambda$ to 1.

Knowledge distillation is a two-step process that starts by freezing the teacher network’s layers and feeding a batch of the input samples from the training images to both networks. Then, it calculates the loss based on the distillation loss defined in the Equation 3 and back-propagate the error only on the student network.

![Figure 2. Number of pruned neurons at each step in different layers of the baseline model. Deeper layers show a higher rate of average zero activation larger than 90% during pruning.](image)

### 4. Experiments

We evaluated our compression method on image-based classification and regression networks. For regression, we compressed a 3D head-pose estimation model trained on the ResNet50 [9] network. For classification, we compressed an image-classification network trained on ResNet110 and ResNet164. We used Keras\(^1\) for implementation.

#### 4.1. Optimizer

We used the stochastic gradient descent (SGD) [1] with momentum 0.09 for training the baseline networks and fine-tuning the pruned baselines and student networks after

\(^1\)https://keras.io/
knowledge distillation. We used Hyperbolic-Tangent Decay (HTD) [11] as our learning-rate scheduler. For fine-tuning the pruned pose-estimation network at each iteration, we used a learning rate of 0.001 for 20 epochs, and for image classification, we used a learning rate of 0.01 for 200 epochs.

The pose-estimation student network was fine-tuned with a learning rate of 0.1 for 100 epochs, and the student image classifier was fine-tuned with a learning rate of 0.001 for 200 epochs. The end learning-rate in HTD scheduler for all experiments was 0.

We used Adam [15] optimizer with an initial learning rate of 0.1 and scheduling of the learning rates to 0.01, 0.001, 0.0001, and 0.00001 after epoch 20, 45, 55, and 65, respectively, for 85 epochs in the knowledge-distillation process.

4.2. Head-pose estimation

The head-pose estimation baseline network was trained on ResNet50 network with two dense layers of size 512 and 128 and a linear layer of size 3 for Yaw, Pitch, and roll estimation at the end of the network. The weights of the network were initialized from the face-recognition network [3]. The input size of the baseline network is 224 × 224 colored images. The baseline was trained on the 300W-LP [33] dataset with image augmentation. All images in the dataset were duplicated and then augmented with random zoom, cropping, and brightness change. For fine-tuning the pruned network at each pruning iteration, we used the same augmented version of 300W-LP and tested on AFLW2000 [33]. The AFLW2000 is a challenging test-set for evaluating head-pose estimation models. It includes ground-truth 3D faces as well as 68 facial-landmarks and head-pose values in yaw, pitch, and roll from the first 2000 images in the AFLW [16] dataset. Unlike the synthesized images in the 300W-LP, the images in the AFLW2000 were gathered from the wild and undergo various illumination and transformations. In addition to testing on AFLW2000, we created three versions of AFLW2000 to test the robustness of the compressed network against motion blur, brightness change, and random cropping of the images as follows:

- **AFLW2000-MB**. Images in the AFLW2000 are convolved with blur kernel of size $25 \times 25$.
- **AFLW2000-LB**. The brightness of the images in the AFLW2000 is reduced with the delta value of $-0.4$.
- **AFLW2000-OC**. Random rectangular black patches of size $35 \times 35$ pixels are applied to the images in the AFLW2000. Table 4 shows 3 different examples from augmented versions of AFLW2000.

4.3. Results

We pruned the prunable layers from the baseline head-pose estimation network for 16 steps. Figure 2 shows the number of pruned neurons for each layer at each pruning step. The results show that the majority of redundancy in the neurons occurs in the first and deeper layers. After fine-tuning the network in the 16-th step, we use it as a teacher network in our knowledge distillation network.
The student network was initialized with random weights and has the same number of filters in each layer as the teacher network. However, the un-prunable (untouched) layers in the student are divided by 32 in all layers in the student network.

We compared the accuracy, the number of parameters, and the final student model’s size with the state-of-the-art head-pose estimation models trained on the same training/testing protocol.

To show the effectiveness of distilling knowledge from teacher to student, we also trained the same student network without distillation. Table 1 shows the accuracy and compression achieved by our method compared to the state-of-the-art. The results show that our compressed pose-estimation model achieves similar results to the state-of-the-art. However, the network in [29] was trained on images of size $64 \times 64$ to achieve a smaller model size.

Since we have compressed the network with the input size of $224 \times 224$ and the larger images represent more information about the subject, the network should perform better against alterations in the input image such as occlusion or motion blur. To make a fair comparison, we tested the augmented version of AFL2000 on our compressed model and state-of-the-art model [29]. Additionally, we tested the student (Student-DS) network on downsampled to $64 \times 64$ version of augmented AFLW2000. Table 2 shows the test results of the student network on the augmented AFLW2000 as well as the downsampled version where our compressed model outperformed [29] in all versions.

| Table 1. Comparisons with state-of-the-art on AFL2000. |
|-----------------|-----|-----|-----|-----|
|                  | Size| Yaw | Pitch| Roll| MAE |
| Dlib (68 points) [14] | -   | 23.1| 13.6| 10.5| 15.8|
| Landmarks [26]       | -   | 5.92| 11.86| 8.27| 8.65|
| 3DDFA [33]           | -   | 5.40| 8.53| 8.25| 7.39|
| Hopenet [26]         | 95.9| 6.47| 6.56| 5.44| 6.16|
| SSR-Net-MD [30]      | 1.1 | 5.14| 7.09| 5.89| 6.01|
| FSA-Caps [29]        | 5.1 | 4.50| 6.08| 4.64| 5.07|
| ResNet-50 (Baseline) | 99.2| 4.38| 4.85| 3.44| 4.22|
| ResNet-50 (Teacher)  | 32.6| 5.38| 5.69| 4.19| 5.03|
| ResNet-50 (Student-Distilled) | 4.9 | 5.89| 5.63| 4.28| 5.25|
| ResNet-50 (Student-Scratch) | 4.9 | 6.95| 6.19| 4.62| 5.90|

4.4. Image Classification

In the second experiment, we used the Cifar10 [17] image-classification dataset. Cifar10 has ten image classes with a total of 50k training and 10k testing images. Here, we trained two ResNet baseline networks (ResNet-110 and ResNet-164) and pruned them for 6 iterations. The pruned networks were used as a teacher in the distillation configuration. The un-prunable layer size was selected in the student network by dividing their size in the teacher network by 2. The prunable layer size between the teacher and the student remained the same. Table 3 shows the compression result on ResNet-110 and ResNet-164. Our method achieved 4.7 and 3.36 compression rates on ResNet-110 and ResNet-164, respectively while the accuracy only dropped by 1%. Additionally, in Table 4 we compare the compression results on ResNet-110 to different compression methods where the results indicate that our method achieves higher compression rate. Note that in FSNET [31], we only compared to pre-quantization results since weight quantization can be applied to all the networks.

Additionally, we compared our compression method with the TensorFlow Model-Optimization Toolkit (TFMO)\(^2\). We test all the models on an ARMv7 processor on a Raspberry Pi2 board.

Table 5 shows a comparison of the baseline model optimized with TFMO and our compression method. While the TFMO archives a $4 \times$ size and a $2 \times$ computation efficiency, the accuracy of the optimized model drops significantly. Our compressed model shows significantly better results, while the estimation accuracy stays close to that of the baseline model.

4.5. Inference-time comparison

We compared the inference speed-up achieved by our model-compression approach with the baseline head-pose estimation model. We also compared the results to FSA-Net

\(^2\)https://www.tensorflow.org/model_optimization
Table 2. Comparisons with state-of-the-art on AFLW2000 with Occlusion, Motion-Blur and Low Brightness.

<table>
<thead>
<tr>
<th></th>
<th>Motion Blur</th>
<th>Low Brightness</th>
<th>Random Occlusion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FSA-Net [29]</td>
<td>Student (ours)</td>
<td>Student-DS (ours)</td>
</tr>
<tr>
<td>Yaw</td>
<td>21.90</td>
<td>12.97</td>
<td>11.56</td>
</tr>
<tr>
<td>Pitch</td>
<td>11.07</td>
<td>8.83</td>
<td>8.73</td>
</tr>
<tr>
<td>Roll</td>
<td>11.05</td>
<td>7.48</td>
<td>7.12</td>
</tr>
<tr>
<td>MAE</td>
<td>14.67</td>
<td>9.76</td>
<td>9.13</td>
</tr>
</tbody>
</table>

Table 3. Compression results on Cifar10

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
<th>#Params</th>
<th>CR Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-110 (Baseline)</td>
<td>94.27</td>
<td>1.74M</td>
<td>-</td>
</tr>
<tr>
<td>ResNet-110 (Teacher)</td>
<td>94.04</td>
<td>0.75M</td>
<td>2.32</td>
</tr>
<tr>
<td>ResNet-110 (Student-Distilled)</td>
<td>93.0</td>
<td>0.37M</td>
<td>4.7</td>
</tr>
<tr>
<td>ResNet-110 (Student-Scratch)</td>
<td>90.0</td>
<td>0.37M</td>
<td>4.7</td>
</tr>
<tr>
<td>ResNet-164 (Baseline)</td>
<td>94.52</td>
<td>2.62M</td>
<td>-</td>
</tr>
<tr>
<td>ResNet-164 (Teacher)</td>
<td>94.30</td>
<td>1.44M</td>
<td>1.81</td>
</tr>
<tr>
<td>ResNet-164 (Student-Distilled)</td>
<td>93.7</td>
<td>0.72M</td>
<td>3.63</td>
</tr>
<tr>
<td>ResNet-164 (Student-Scratch)</td>
<td>89.6</td>
<td>0.72M</td>
<td>3.63</td>
</tr>
</tbody>
</table>

Table 4. Cifar10 compression comparison on state-of-the-art

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
<th>#Params</th>
<th>CR Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-110-Student (Ours)</td>
<td>93.0</td>
<td>0.37M</td>
<td>4.7</td>
</tr>
<tr>
<td>ResNet-110- Filter pruning [21]</td>
<td>93.30</td>
<td>1.16M</td>
<td>1.5</td>
</tr>
<tr>
<td>ResNet-110- FSNET [31]</td>
<td>93.81</td>
<td>0.44M</td>
<td>3.97</td>
</tr>
</tbody>
</table>

Table 5. Comparison between baseline model compressed with our method and TFMO.

<table>
<thead>
<tr>
<th>Model name</th>
<th>MAE</th>
<th>Size (MB)</th>
<th>Inference (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-50 (Baseline)</td>
<td>4.77</td>
<td>98 MB</td>
<td>2.5</td>
</tr>
<tr>
<td>ResNet-50 (TFMO)</td>
<td>24.4</td>
<td>24.7 MB</td>
<td>1.3</td>
</tr>
<tr>
<td>ResNet-50 (Student)(Ours)</td>
<td>5.25</td>
<td>4.9 MB</td>
<td>0.4</td>
</tr>
</tbody>
</table>

5. Conclusion

In this paper, we proposed a compression method for network architectures such as ResNets. First, we showed that the ResNet networks’ neurons could be pruned only on specific layers. Second, we showed that our knowledge-distillation architecture and loss function reduce the number of weights that could not be pruned due to dimensionality dependencies. Our compression approach works on ResNet-based regression and classification networks.

We used ResNet50 head-pose estimation network for regression tasks. For classification, we used ResNet-110 and ResNet-164. While we achieved similar results in the head-pose estimation network to the state-of-the-art on both model size and accuracy, we showed that our compressed head-pose estimation model outperforms the state-of-the-art on heavily occluded test images. In image classification networks, we achieved similar results to the state-of-the-art accuracy but outperformed on the compression rate.

Finally, as the primary goal of compression is real-time inference on low-end systems, we compared the inference time on multiple embedded GPU and CPU boards. The results presented in this work show the possibility of deploying large Convolutional Neural Networks in low-cost embedded computers for real-world application in the industry.

In this paper, we focused mainly on the ResNet architecture. In future work, we plan to apply the method to different architectures, and also on networks trained on larger image datasets such as Cifar100 [17] and ImageNet [18].

References


[28] Han Vanholder. Efficient inference with tensorrt, 2016. 6


