Università della Svizzera italiana

Faculty of Informatics

USI Technical Report Series in Informatics

Pruning Highway Networks For Reducing Memory Usage

Dhananjay Tomar[1]

[1] Faculty of Informatics, Università della Svizzera italiana, Switzerland

Abstract

In recent years, deep learning has gained a lot of popularity for achieving state-of-the-art results in various tasks like image classification, face recognition, machine translation, speech recognition, etc. This success, however, has come with models getting bigger in size. As they get bigger, they also get computationally more expensive. For example, the ResNet-152 model which won the ILSVRC 2015 image classification task has a size of about 240 MB and requires 11.3 billion FLOPS to classify one image. This prevents them from being used in mobile devices which tend to have much less memory and computational resources. In this work, we propose a novel approach to reduce the size of highway networks (a type of neural network architecture which has been shown to work well in practice) by pruning their nodes. We test our approach on a highway network trained on the CIFAR-10 dataset. Our approach results in reducing the size of the network by 85% (34 MB to 5.2 MB) while incurring a 1.72% loss in accuracy.

Report Info

Published
July 2017

Number
USI-INF-TR-2017-5

Institution
Faculty of Informatics
Università della Svizzera italiana
Lugano, Switzerland

Online Access
www.inf.usi.ch/techreports

1 Introduction

Deep neural networks prove to be both computationally intensive and memory intensive which makes them difficult to deploy on mobile devices with limited hardware resources. The size of ResNet-152 being 240 MB or VGG-16 being 552 MB doesn't sound very big compared to other things we store in embedded systems but the difference is that neural networks reside in RAM while running. Even if it was feasible to have large models in some embedded systems, power consumption would still be an issue as these large models need to perform billions of FLOPS for a single input. This calls for a solution which can significantly reduce memory and power consumption.

One solution to this problem is to store the neural network on a cloud and run it there, providing it as a service. But this solution comes with its own problems, some of which are following:

- **Bandwidth Issues**: In order to use that service, the user will have to send his/her data to the server. The type of data sent will vary depending on the task being performed by the neural network. For example, the user might send images for neural network to detect and tag faces or an audio clip for the neural network to transcribe it or some text to translate it to another language, etc. If the data being sent is not very small then the user will have to rely on the bandwidth for a fast response and even if the data is not big in size, connectivity could still be a problem.
- **Data Privacy and Security**: When the user sends some data from his/her device to a server, privacy and security issues naturally arise. The service will have to take care of keeping the user data private and making sure that it is securely transmitted to and from the mobile device.
- **Monetary Costs**: Depending on the number of users, running neural networks in the cloud can prove to be very costly. If the neural network is large and requires lots of computation, running it on a

CPU will prove to be very slow, therefore, it'll be run on a GPU as is usually done. Depending on the number of requests the service gets, running multiple copies of neural networks might be required which translates to possibly using multiple GPUs (depending on the memory available in a GPU) and thus the costs can be very high.

The other solution is to reduce the size of the neural network so that it can easily fit in the memory, needs to perform fewer FLOPS and hence also uses less power. In this work, we reduce the size of a highway network (discussed below) by pruning its nodes. In contrast to pruning weights, removing nodes results in weight matrices that are still dense. Therefore, the neural network doesn't have to rely on libraries which provide operations like convolution and matrix-matrix multiplication for sparse matrices. Existing efficient libraries for dense matrices can still be used.

1.1 Highway Networks

The increasing depth of neural networks over the years has been one of the main reasons for their success. But training deep networks proves to be challenging — beyond a certain depth, networks start losing performance instead of gaining it. Highway networks [3] were the first networks to overcome this issue. Highway networks don't lose performance even when they are very deep. They have been shown to work well in practice on various tasks [1].

A standard neural network layer performs a transformation H on input x using its parameters θ_H — weights W_H and biases b_H; followed by a non-linear activation function. H is usually an affine transformation or a convolution operation. The output y of a standard layer is then given by:

$$y = H(x, \theta_H)$$

A highway layer besides calculating H, performs two other transformations C and T. The output y of the highway layer is given by:

$$y = H(x, \theta_H).T(x, \theta_T) + x.C(x, \theta_C)$$

where . represents dot product. C in the above equation is called the carry gate and T is called the transform gate because C dictates how much information from the previous layer is going to be carried forward and T dictates how much of the transformed information is going to be used. In practice, C and T are always coupled by setting

$$C = 1 - T$$

which implies that the output y of the highway layer is:

$$y = H(x, \theta_H).T(x, \theta_T) + x.(1 - T(x, \theta_T)) \tag{1}$$

Note that in Equation 1, the dimensions of x, H and T must be same. The activation function used for T is sigmoid given by $\sigma(x) = \frac{1}{1+e^{-x}}$ whose output is between 0 and 1. One of the reasons for using this activation function is because of its interpretation as gate being closed when output is close to 0 or it being open when output is close to 1. This gating mechanism in highway networks is inspired by the gating mechanism in LSTM which also uses sigmoid as the activation function for its gates.Figure 1 shows a highway layer.

2 Architecture Used

The highway network used in this work is a convolutional neural network. We slightly modify the architecture from "Wide Residual Networks" [5]. To define our architecture we use following notations:

- **Convolutional Layer**: A convolutional layer is denoted as

  $$\text{Conv(filter height} \times \text{filter width, Number of filters, Stride)}$$

  Note that we use same stride for both axes — along height and width.

- **Highway Layer**: The same notation is used for a highway layer, i.e.

  $$\text{Highway(filter height} \times \text{filter width, Number of filters, Stride)}$$

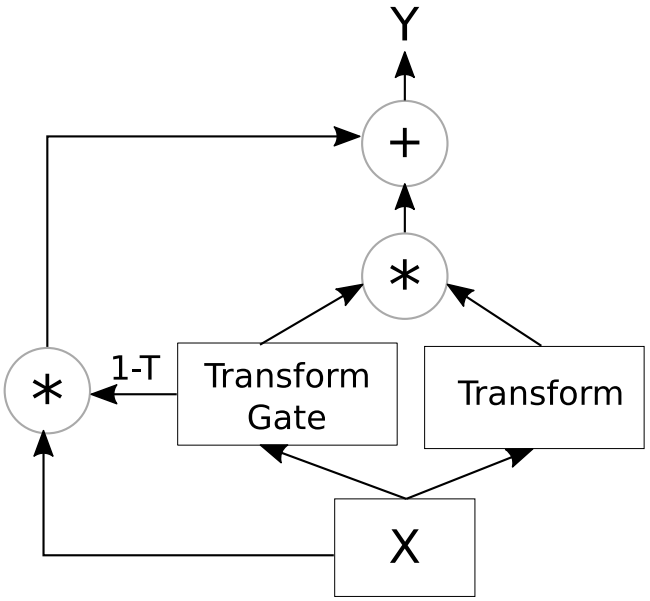


Figure 1: A highway layer with coupled gates where T represents the output of the transform gate and '*' represents a dot product. 'Transform' refers to H in Equation 1.

but we slightly modify the standard highway layer by using the output of H as input for T. The output of the highway layer is then given by:

$$y = H(x, \theta_H).T(H(x, \theta_H), \theta_T) + x.(1 - T(H(x, \theta_H), \theta_T))$$

We found this to work equally well and it comes with an added advantage that it helps in reducing the number of parameters in the highway layer when we prune it by reducing the number of feature maps (outputs of filters) that T has to operate on.

To denote multiple layers which have same architecture and are stacked on top of each other, we denote them as: Highway $(\ldots)\times$ Number of layers.

The architecture used can be described as

$$\begin{array}{c}
\text{Conv}\,(3\times 3, 16, 1) \\
\downarrow \\
\text{Highway}\,(3\times 3, 64, 1) \\
\downarrow \\
\text{Highway}\,(3\times 3, 64, 1)\times 5 \\
\downarrow \\
\text{Highway}\,(3\times 3, 128, 2) \\
\downarrow \\
\text{Highway}\,(3\times 3, 128, 1)\times 5 \\
\downarrow \\
\text{Highway}\,(3\times 3, 256, 2) \\
\downarrow \\
\text{Highway}\,(3\times 3, 256, 1)\times 5 \\
\downarrow \\
\text{Global average pooling} \\
\downarrow \\
\text{Fully Connected Layer}
\end{array}$$

3 Dataset Used

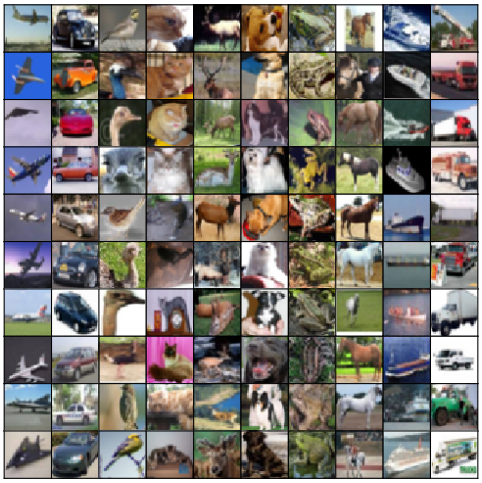


Figure 2: Each column shows ten randomly chosen images from a class. There are ten columns — one for each class

We use CIFAR-10 dataset [2] for this work. CIFAR-10 is one of the most popularly used datasets. CIFAR-10 consists of 60,000 coloured (3 channels — red, green and blue) images of size 32×32. The dataset is divided into 5 train batches and 1 test batch, each having 10,000 images. The test set has 1000 images from each class, the train batches have random images in them so each batch has more images from one class than another.

Each image belongs to one of the 10 classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck. The classes are totally mutually exclusive. There is no overlap between automobiles and trucks. Each class has 6000 images in it (some of which are present in the test set). Figure 2 some random images from CIFAR-10.

4 Method

In this section we describe the method used to prune the highway networks. We start by describing the interpretation of transform gates which helps us to understand how they can be used to judge which nodes to prune. Then we discuss how nodes are pruned.

4.1 Transform Gate

The transform gate regulates the flow of information in a highway network. In the extreme cases, if the transform gate is closed i.e. its value is 0 then the output of the highway layer is equal to its input i.e. $y = x$; on the other hand, if the transform gate is open i.e. its value is 1 then the output of highway layer is $y = H(x, \theta_H)$. So, the closer to 0 the value of the transform gate is, the less the output of H flows forward (to the next layer) or in other words, the less is the output used.

4.2 Pruning the nodes

We prune a node if its output is never used much. If the output of a node is not used for any example in the training data i.e. the value of the corresponding transform gate is close to 0 for every single example in the training data then the node can be pruned and along with it the corresponding transform gate as its output is no longer needed for the pruned node and $1 - T$ can be assumed to be 1 for small T.

When highway networks are trained in the standard way then the outputs of transform gates are not close to 0 for all the inputs. To bring the values of transform gates close to 0, we can incentivize the network to do so. In this work we incentivize it by adding an additional term to the loss function. To understand how the additional cost term is calculated, we need to know under what conditions is the filter pruned. The following conditions should be met for a filter to be pruned:

- The value of every single element in the transform gate's feature map (the output of the transform gate's filter) should be below a certain chosen threshold. In other words, the element with the maximum value in the feature map should be below the threshold. This motivates us to decrease the value of the element with maximum value in the feature map with our cost term.
- The feature map should meet the condition in the previous point for every single example in the training data. Let's call the value of the element with the maximum value in the feature map as "feature map value". Then the feature map value of the example with the maximum feature map value should be below the threshold. The motivates us to have an objective function which brings down the the feature map values of the example with the maximum feature map values.

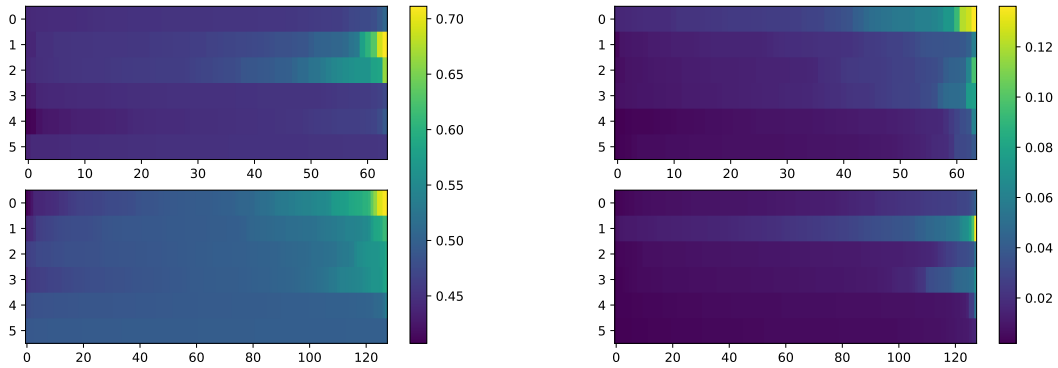
For a mini-batch having b training examples, the additional cost term is calculated as follows:

1. In each feature map, find the maximum value i.e. the feature map value.
2. For each example in the batch, calculate the sum of the feature map values. Let's call this value sum_i for an example with index i in the batch.
3. The additional cost term for the batch is the sum of the example with maximum sum or simply $\max_i(\text{sum}_i)$, where $i \epsilon [1, b]$.

The network is trained with this additional cost term added to the loss function, until the solution converges to a local minima. We then prune the filters in the network as described above for a chosen threshold value. Once the network has been pruned, we train the pruned network further for 20 epochs and report the highest accuracy attained on the validation dataset during these 20 epochs.

5 Results

For the architecture used in this work, we observed that the accuracy of the network was a little (< 1%) less when it was trained with additional cost term (described above) added to its loss function. However, the output values of the transform gates decreased significantly – the average value of sum_i decreased from about 1400 to 41. Figure 3 shows the maximum values of feature maps for the first 12 layers in both incentivized and unincentivized networks. Notice the different ranges in the colorbars of the plots.



(a) Unincentivized network

(b) Incentivized network

Figure 3: This figure shows heat maps for maximum values of feature maps where maximum is calculated over the whole feature map (maximum value of the feature map) and over all training examples. Each row shows a layer of the network. The feature maps for each layer have been sorted according to their values.

This shows that highway networks can learn to not use the output of $H(x, \theta_H)$ and rely mostly on the information available from the input x which means that there is a lot of room for pruning nodes (filters) in the highway networks.

Pruning the network requires setting a threshold value. The filters whose corresponding transform gates filters' output values are all below this threshold for all the examples in the training data are pruned.Figure 4

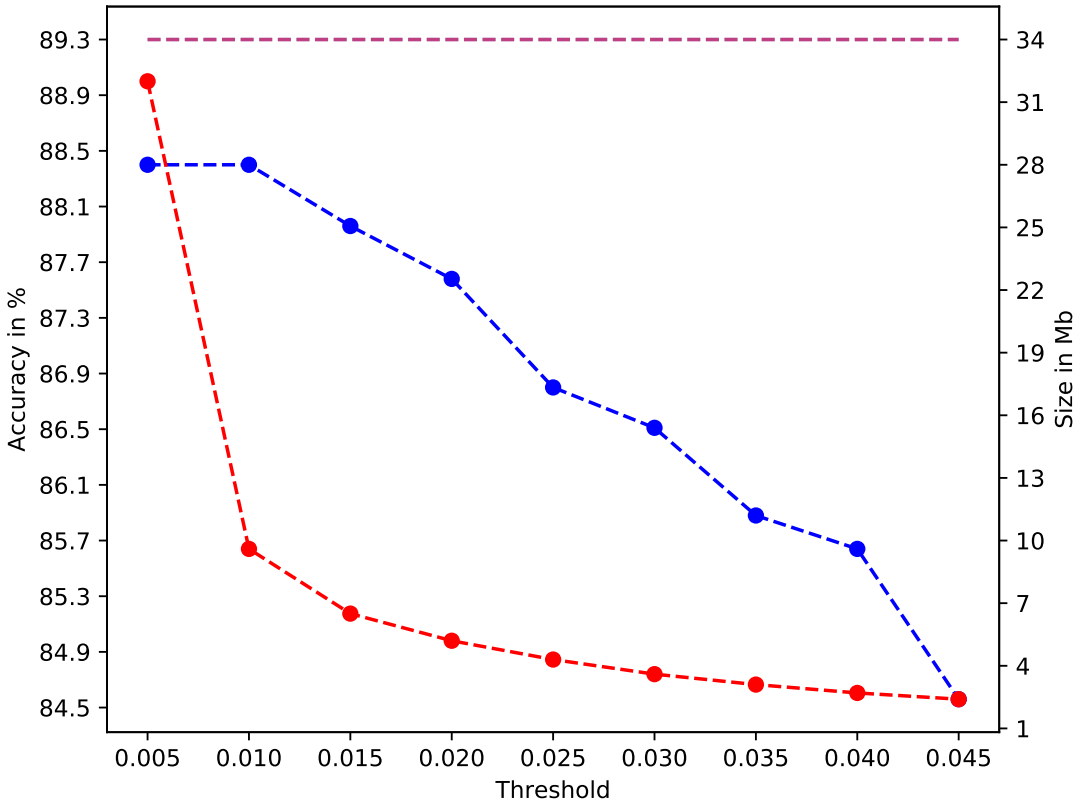


Figure 4: Accuracy (blue) and size (red) of the network for different threshold value. The horizontal line at the top shows the accuracy and model size of the network trained without any incentive to use fewer transformations i.e. the additional cost term was not added to the loss function.

shows results for different threshold values — the blue line shows accuracy of the model and the red line shows model's size.

As we can we in Figure 4 the size of the network drops significantly from being 32 MB (close to original model size) to 9.6 MB when the threshold is increased from 0.005 to 0.010. This means that the maximum value a filter can have lies somewhere between this range for a large percentage of filters. And while the model size drops significantly, the accuracy on validation set does not drop at all — for both the thresholds (0.005 and 0.010), it is 88.4%. The accuracy however starts declining after 0.010 along with the model size. For the threshold value of 0.045, the accuracy is 84.56% with model size being 2.4 MB. This is a bit surprising because the threshold value is not high.

The value of the transform gate can be interpreted as percentage of information used from the transform $H(x, \theta_H)$ and the remaining from the input x as it lies in a certain range (0 to 1) with 0 being 0% information used from H and 1 being 100% information used from H. Therefore when we choose a threshold value ζ for the transform gate we are basically deciding that if only $(\zeta \times 100)$% of information is being used from H, ignore it. In Figure 4 we have varied this percentage from 0.5% to 4.5%. And from these experiments, it seems like even a small percentage of information like 4% is necessary to maintain the accuracy which needs to be explored in future work.

One reason for so much loss in accuracy could be that the layers with stride 2 in our architecture shuold not be pruned. It has been shown that these layers where input gets downsampled (decrease in the size of feature maps) can prove to be crucial [4]. We therefore ran the same experiment again but this time did not prune the downsampling layers. Figure 5 shows results for this experiment. As we can see, the accuracy still drops significantly thus falsifying the hypothesis that pruning the downsampling layers is not the reason for loss in accuracy. Note that the accuracies in this experiment are to be higher than in previous experiment for higher threshold values but it's not a fair comparison as the model sizes in this experiment are also higher (because we are not reducing the size of downsampling layers).

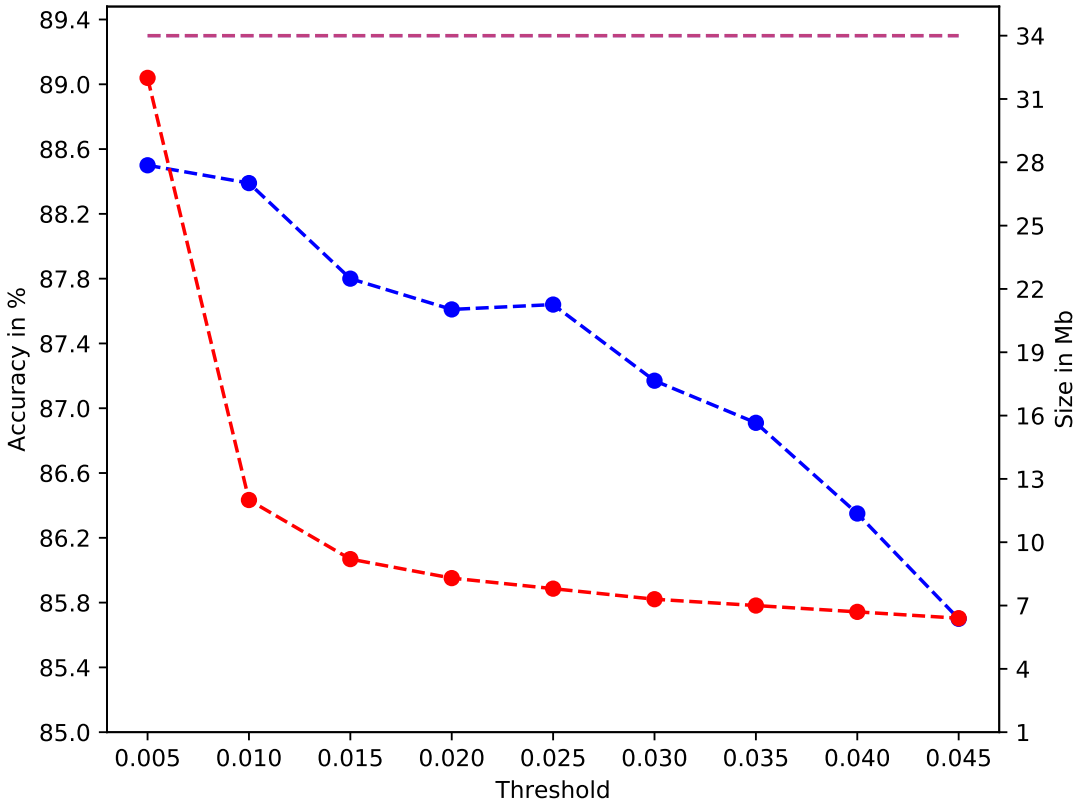


Figure 5: Accuracy (blue) and size (red) of the network for different threshold value. The horizontal line at the top shows the accuracy and model size of the network trained without any incentive to use fewer transformations i.e. the additional cost term was not added to the loss function. The downsampling layers were not pruned for any threshold value.

6 Conclusion and Future Work

In this work, we proposed a novel way of pruning nodes/filters in highway networks and analyzed its performance. We found that even a small percentage of information used from the output of the filters in H can prove to be crucial as pruning those filters can result in loss in accuracy. We then hypothesized a possible reason for such behaviour but found it to not hold true.

The proposed approach seems to be orthogonal to previously proposed approaches but it needs to be empirically verified in future work by verifying that pruning the network further using the previously proposed approaches does not result in loss in accuracy. The reason for loss in accuracy for seemingly small threshold values should be explored in future work as it might be due to some problem which when solved could result in even more reduction in the network's size.

References

[1] Klaus Greff, Rupesh K Srivastava, and Jürgen Schmidhuber. Highway and residual networks learn unrolled iterative estimation. *arXiv preprint arXiv:1612.07771*, 2016.

[2] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

[3] Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber. Training very deep networks. In *Advances in neural information processing systems*, pages 2377–2385, 2015.

[4] Andreas Veit, Michael J Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks. In *Advances in Neural Information Processing Systems*, pages 550–558, 2016.

[5] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.